中等职业教育国家规划教材
全国中等职业教育教材审定委员会审定

C 语言编程基础
（第 2 版）

王 森 主 编

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本版仍保留了第一版的基本结构和风格，全书分成基础模块和选修模块两篇共 10 章。其主要内容有：C 语言的基本概念、主要特点、Turbo C 集成环境的使用；C 语言的数据类型、C 程序结构组成与设计方法；C 语言函数的分类与调用、数组、文件与指针操作等。但本次再版做了不少改动和更新，特别是对各章的例题、习题、上机实训内容进行了修订、更新和扩充，使本书更加突出了实用性、科学性和先进性。

本书针对中等职业教育层次，难易适中、实用性强、便于接受，除了可作为中等职业学校 3、4 年制计算机及应用专业的专业教材外，还可作为其他专业的选修教材及一般计算机爱好者的自学参考书。

本书配有电子教学资料包，内容包括教学指南、电子教案及习题答案，详见前言。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

C 语言编程基础 / 王森主编.—2 版.—北京：电子工业出版社，2008.8
中等职业教育国家规划教材
ISBN 978-7-121-06963-5

I. C… Ⅱ. 王… Ⅲ. C 语言－程序设计－专业学校－教材 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2008）第 091492 号

策划编辑：关雅莉
责任编辑：关雅莉 肖博爱
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×1 092　1/16　印张：14.25　字数：364.8 千字
印　　次：2008 年 8 月第 1 次印刷
印　　数：7 000 册　　定价：19.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。服务热线：（010）88258888。

前　言

本书第一版是根据教育部颁发的《编程语言基础》课程教学大纲于 2002 年初编写发行的，适合中等职业学校计算机及应用专业的学生使用。出版后被多个省、市的中等职业学校采用，除作为专业教材外，还被许多相近专业选作为选修课教材，并受到了 C 语言爱好者和初学者的普遍欢迎，使用效果良好，发行量较大。几年来，出版社和作者通过多次与各地的一线教师交流经验，汲取了许多好的意见和建议，再加上 IT 行业形势的不断发展和教学环境的不断改善，以及作者的教学实践体会，本次再版做了不少改动和更新，使本书更加突出了实用性、科学性和先进性。但本书仍保留了原书的基本结构和风格，如：紧扣教学大纲，在教学内容安排上采用模块化、单元化的结构；在叙述方法上采用由浅入深、由易到难的原则(且尽量将难点分散)；在实践安排上增加上机单元，突出技能训练。在每一部分内容中，都有基本概念介绍、知识运用举例、编程及算法分析、思考练习题、上机指导与实训等。这样安排，使学生更容易理解、消化、掌握各单元、各模块的学习内容。

本次修订，突出了以下几个方面：对第一版中的某些概念进行了重新归类；有的章节在标题、组织结构、以及行文等方面都做了较大的改动；对各章的例题、习题、上机实训内容进行了修订、更新和扩充。全书内容和参考学时安排如下表所示：

分　类	序　号	课 程 内 容	学 时 数
基础知识模块	1	C 语言概述	2+(2)
	2	基本数据类型和简单程序设计	6
	3	表达式与运算符	3+(2)
	4	数据的输入与输出	5
	5	控制结构程序设计	16
	6	数组	8+(2)
	7	函数	10
	8	文件	4+(2)
可选模块	9	编译预处理	2
	10	指针	8
机　动			8
合　计			27

本书大纲给出的参考教学时数为 72 学时，其中授课时间为 44 学时，上机实习 20 学时，机动为 8 学时。根据各章内容的多寡，我们在各章中给出了课时安排建议供备课时参考。机动学时基本想法是加在上机实习上，有的讲授内容可以结合上机进行讲解。

本书的编写和出版工作由电子工业出版社组织实施，王森教授担任主编和统稿，并编写了第 2、4、5、8 章及附录，成都职业技术学院向华老师编写了第 1、6、7 章，山东信息职业技术

学院季昌武、韩凤文老师编写了第 3、9、10 章。在本书编写过程中，得到了出版社和同行老师们的支持与帮助，在此一并表示诚挚的感谢！由于作者水平有限，书中难免存有缺点和错误，殷切希望广大读者批评指正。

为了方便教师教学，本书还配有教学指南、电子教案及习题答案（电子版）。请有此需要的教师登录华信教育资源网（www.huaxin.edu.cn 或 www.hxedu.com.cn）免费注册后再进行下载，有问题时请在网站留言板留言或与电子工业出版社联系（E-mail:hxedu@phei.com.cn）。

验证码（资料包下载密码）使用说明

本书封底验证码为配套资料包下载密码。

下载电子教学参考资料包前请登录华信教育资源网（公共网：www.hxedu.com.cn教育网：www.huaxin.edu.cn），免费注册成为网站的会员。注册并激活会员账户成功后，请先用注册用户在网站登录，然后用本书书名或作者名检索本书，单击进入本书终极页面，您会看到本书配套电子教学参考资料包，单击“下载”按钮，会弹出资料包下载密码输入框，请输入封底标签上的验证码，验证通过后即可下载。下载时请勿使用网际快车或迅雷等下载工具。资料包下载密码只能使用一次，逾次作废。

本书验证码在资料包下载时能够验证通过，则说明本书为正版图书。

使用本书验证码下载资料包时如有任何问题，请拨打电话 010-88254485 或发邮件至 hxedu@phei. com.cn。

编　者

2008 年 8 月

中等职业教育国家规划教材出版说明

为了贯彻《中共中央国务院关于深化教育改革全面推进素质教育的决定》精神，落实《面向21世纪教育振兴行动计划》中提出的职业教育课程改革和教材建设规划，根据《中等职业教育国家规划教材申报、立项及管理意见》（教职成［2001］1号）的精神，教育部组织力量对实现中等职业教育培养目标和保证基本教学规格起保障作用的德育课程、文化基础课程、专业技术基础课程和80个重点建设专业主干课程的教材进行了规划和编写，从2001年秋季开学起，国家规划教材将陆续提供给各类中等职业学校选用。

国家规划教材是根据教育部最新颁发的德育课程、文化基础课程、专业技术基础课程和80个重点建设专业主干课程的教学大纲（课程教学基本要求）编写的，并且经全国中等职业教育教材审定委员会审定。新教材全面贯彻素质教育思想，从社会发展对高素质劳动者和中初级专门人才需要的实际出发，注重对学生的创新精神和实践能力的培养。新教材在理论体系、组织结构和阐述方法等方面均进行了一些新的尝试。新教材实行一纲多本，努力为学校选用教材提供比较和选择，满足不同学制、不同专业和不同办学条件的学校的教学需要。

希望各地、各部门积极推广和选用国家规划教材，并且在使用过程中，注意总结经验，及时提出修改意见和建议，使之不断完善和提高。

教育部职业教育与成人教育司

2001年10月

目　录

第一篇　基础知识模块

第 1 章　C 语言概述……3
1.1　C 语言简史及特点……3
1.1.1　C 语言的发展……3
1.1.2　C 语言的特点……4
1.2　C 语言程序……4
1.2.1　几个典型的 C 程序……4
1.2.2　C 程序的基本结构……6
1.2.3　C 语言的基本符号与词汇……6
1.3　C 语言集成开发环境……8
1.3.1　Turbo C 集成开发环境介绍……8
1.3.2　Turbo C 集成开发环境的使用……8
习题一……11
上机实习指导……12
上机实习　Turbo C 的基本操作……13
第 2 章　数据类型和简单程序设计……16
2.1　基本知识……16
2.1.1　C 语言的数据类型……16
2.1.2　常量与变量的使用……16
2.1.3　C 语句……17
2.1.4　结构化程序设计方法……18
2.2　整型数据……19
2.2.1　整型常量……19
2.2.2　整型变量及分类……19
2.2.3　整数运算的程序设计……21
2.3　浮点型数据……22
2.3.1　浮点常数……22
2.3.2　浮点型变量与分类……22
2.3.3　符点数运算的程序设计……23
2.4　字符型数据……24
2.4.1　字符常数……24

2.4.2 字符变量及分类……25
2.4.3 字符串……26
2.4.4 最简单的字符和字符串处理程序……27
2.5 类型的混合运算……28
2.5.1 类型的自动转换……28
2.5.2 类型的强制转换……29
习题二……30
上机实习指导……32
上机实习 基本数据类型的简单程序设计……33
第 3 章 表达式与运算符……35
3.1 概述……35
3.1.1 表达式……35
3.1.2 运算符……36
3.2 算术运算符与算术表达式……36
3.2.1 算术运算符……36
3.2.2 算术表达式……37
3.3 其他运算符的应用……39
3.3.1 赋值运算符和赋值表达式……39
3.3.2 自增和自减运算符……40
3.3.3 条件运算符和条件表达式……41
3.3.4 逗号运算符和逗号表达式……43
3.3.5 sizeof 运算符……44
3.4 运算符的优先级与结合性……44
习题三……45
上机实习指导……46
上机实习 运算符及表达式应用……47
第 4 章 数据的输入与输出……49
4.1 字符输入/输出函数……49
4.1.1 字符输入函数 getchar()……49
4.1.2 字符输出函数 putchar()……50
4.1.3 字符的连续输入/输出……51
4.2 按格式输入/输出函数……52
4.2.1 按格式输出函数 printf()……52
4.2.2 按格式输入函数 scanf()……57
习题四……62
上机实习指导……64
上机实习一 字符的输入/输出……65
上机实习二 格式输入/输出函数的使用……67
第 5 章 控制结构程序设计……69
5.1 关系表达式与逻辑表达式……69

5.1.1 关系运算符与关系表达式 ……69
5.1.2 逻辑运算符与逻辑表达式 ……70
5.2 条件选择语句 if ……71
5.2.1 if 语句的三种形式 ……71
5.2.2 条件分支程序设计举例 ……73
5.3 开关语句 switch ……75
5.3.1 switch 语句结构形式 ……75
5.3.2 多路选择程序设计举例 ……77
5.4 循环语句 ……79
5.4.1 for 语句 ……79
5.4.2 while 语句 ……83
5.4.3 do-while 语句 ……85
5.5 中止语句 break 和继续语句 continue ……87
5.5.1 break 语句 ……87
5.5.2 continue 语句 ……88
5.6 无条件转移语句 goto ……88
5.6.1 goto 语句的格式 ……88
5.6.2 goto 语句应用举例 ……89
习题五 ……89
上机实习指导 ……91
上机实习一 关系表达式和逻辑表达式 ……91
上机实习二 if 语句 ……93
上机实习三 switch 语句 ……94
上机实习四 for 语句 ……96
上机实习五 while 语句和 do-while 语句 ……97
第 6 章 数组 ……99
6.1 一维数组的定义和引用 ……99
6.1.1 一维数组的定义 ……99
6.1.2 一维数组的引用 ……100
6.1.3 一维数组的初始化 ……103
6.1.4 一维数组应用举例 ……103
6.2 二维数组的定义和引用 ……105
6.2.1 二维数组的定义 ……105
6.2.2 二维数组的引用 ……107
6.2.3 二维数组的初始化 ……107
6.2.4 二维数组应用举例 ……109
6.3 字符数组 ……109
6.3.1 字符数组的定义和初始化 ……110
6.3.2 字符数组的引用 ……110
6.3.3 字符串与字符数组 ……111

6.3.4 字符串的输入、输出和处理……111
习题六……116
上机实习指导……117
上机实习一 数值型数组……119
上机实习二 字符型数组……121
第 7 章 函数……124
7.1 函数概述……124
7.1.1 为什么要使用函数……124
7.1.2 库函数和用户自定义函数……125
7.2 函数的定义……125
7.2.1 函数定义的一般形式……125
7.2.2 有关函数定义的几点说明……126
7.3 函数参数及返回值……127
7.3.1 函数参数……127
7.3.2 函数的返回值……129
7.4 函数的调用……130
7.4.1 函数的语句调用……130
7.4.2 函数表达式调用……131
7.4.3 函数的递归调用……133
7.4.4 函数的嵌套调用……135
7.5 数组作函数参数……136
7.5.1 数组元素作函数参数……136
7.5.2 数组名作函数参数……136
7.6 变量的作用域和生存期……138
7.6.1 变量的作用域……138
7.6.2 变量的生存期……141
7.7 函数的作用域……143
7.7.1 内部函数……143
7.7.2 外部函数……143
习题七……144
上机实习指导……147
上机实习一 函数的定义和调用……148
上机实习二 局部变量和全局变量……150
上机实习三 一个应用程序的设计……152
第 8 章 文件……156
8.1 C 语言文件概述……156
8.1.1 C 语言文件的存储方式及分类……156
8.1.2 文件指针……157
8.2 文件的打开与关闭……157
8.2.1 文件的打开（fopen（）函数）……157

8.2.2 文件的关闭（fclose()函数）……159
8.3 文件的读写……159
8.3.1 字符的输入和输出……159
8.3.2 字符串的输入和输出……161
8.3.3 按格式输入和输出……162
8.4 随机文件的读写……164
8.4.1 fseek()函数……164
8.4.2 fread()与 fwrite()函数……164
习题八……166
上机实习指导……167
上机实习一 文件的顺序读写……168
上机实习二 文件的随机读写……170

第二篇 选用模块

第 9 章 编译预处理……175
9.1 宏定义……175
9.1.1 不带参数的宏定义……175
9.1.2 带参数的宏定义……176
9.2 文件包含……178
9.2.1 使用格式……178
9.2.2 使用说明……179
9.3 条件编译……179
9.3.1 形式 1……179
9.3.2 形式 2……180
9.3.3 形式 3……180
习题九……181
第 10 章 指针……184
10.1 指针概述……184
10.1.1 变量的地址与指针变量……184
10.1.2 指针变量的定义……185
10.1.3 指针变量的操作……186
10.2 指针与数组……188
10.2.1 一维数组的指针……188
10.2.2 二维数组的指针……190
10.2.3 指向字符串的指针……192
10.3 指针与函数……193
10.3.1 指针作为函数的参数……193
10.3.2 返回指针值的函数……196
10.3.3 指向函数的指针……197

习题十……199
上机实习指导……201
上机实习　指针的简单应用……202
附录 A　ASCII 码字符表……204
附录 B　Turbo C 库函数简介……205
附录 C　Turbo C 编译错误信息……213

第一篇　基础知识模块

第一篇　基础知识模块

基础知识模块是中等职业学校计算机及应用专业学生必须掌握的知识，其教学目标是:

1. 理解编程语言的基本概念，具有使用 C 语言编程的能力;
2. 掌握结构化程序设计的方法和技巧;
3. 掌握 C 语言的基本语法、基本符号、词汇等;
4. 掌握数据类型、函数、语句的基本知识及其应用;
5. 理解编程语言有关算法的思想;
6. 掌握数组知识和使用方法;
7. 初步掌握文件的使用方法;
8. 具有阅读程序的能力并掌握上机调试程序的方法。

同时在教学中，注意锻炼学生的逻辑思维能力；教育学生具有严谨的学风、创新意识和创新精神、科学的求学态度，以及互助合作的团队精神。

第 1 章　C语言概述

C 语言是编程语言中较为流行的一种。随着计算机的普及和发展，C 语言在各个领域的应用越来越广泛。几乎各类计算机都支持 C 语言的开发环境，这对 C 语言的普及及应用奠定了基础。本章主要介绍 C 程序的结构、C 语言的基本语法、C 语言集成开发环境等 C 语言的基本知识。

【本章要点】

1. C 语言的发展及特点；
2. C 程序的基本结构；
3. C 语言的基本符号与词汇；
4. C 语言集成开发环境。

【学习目标】

1. 掌握 C 程序的基本结构；
2. 掌握 C 语言的基本符号与词汇；
3. 了解 Turbo C 集成开发环境的基本使用方法；
4. 熟悉在 Turbo C 中编辑和运行最简单的 C 程序。

【课时建议】

讲授 2 课时，上机 2 课时（利用机动课时）。

1.1　C语言简史及特点

1.1.1　C语言的发展

C 语言是一种编译性程序设计语言，它与 UNIX 操作系统紧密地联系在一起。UNIX 系统是通用的、交互式的计算机操作系统，它诞生于 1969 年，是由美国贝尔实验室的 K.Thompson 和 D.M.Ritchie 用汇编语言开发成功的。

C 语言的前身是 BCPL 语言。1967 年英国剑桥大学的 Martin Richard 推出 BCPL 语言（Basic Combined Programming Language）。1970 年贝尔实验室的 K.Thompson 以 BCPL 语言为基础，开发了 B 语言，并用 B 语言编写了 UNIX 操作系统，在 PDP-7 计算机上实现。1972 年贝尔实验室的 D.M.Ritchie 在 B 语言的基础上设计出 C 语言，C 语言既保持了 BCPL 语言和 B 语言的精练、接近硬件的优点，又克服了它们过于简单的缺点。1973 年，K.Thompson 和 D.M.Ritchie 合作把 UNIX 的 90%以上用 C 语言改写，并加进了多道程序设计的功能，称为 UNIX 第五版，开创了 UNIX 系统发展的新局面。1975 年 UNIX 第六版颁布后，C 语言得到计算机界的普遍认可，从此，C 语言与 UNIX 系统一起互相促进，迅速发展。

最初，设计 C 语言的目的只是为了描述和实现 UNIX 操作系统。目前，C 语言已独立于 UNIX

系统，先后被移植到大、中、小型计算机及微机上。1978 年 B.Kernighan 和 D.M.Ritchie 合作编写了经典著作“The C Programming Language”，它是目前所有 C 语言版本的基础。1983 年美国国家标准化协会（ANSI）对 C 语言问世以来的各种版本进行了扩充，制定了 ANSI C。现在流行的 C 语言版本有：Microsoft C，Turbo C，Quick C，Borland C 等。本书主要介绍 Turbo C。

1.1.2 C语言的特点

C 语言具有以下几个基本特点：

（1）C 语言是结构化程序设计语言。C 语言程序的逻辑结构可以用顺序、选择和循环三种基本结构组成，便于采用自顶向下、逐步细化的结构化程序设计技术。用 C 语言编制的程序，具有容易理解、便于维护的优点。

（2）C 语言是模块化程序设计语言。C 语言的函数结构、程序模块间的相互调用及数据传递和数据共享技术，为大型软件设计的模块化分解技术，以及软件工程技术的应用提供了强有力的支持。

（3）C 语言具有丰富的运算能力。C 语言除了具有一般高级语言所拥有的四则运算及逻辑运算功能外，还具有二进制的位（bit）运算、单项运算和复合运算等功能。

（4）C 语言具有丰富的数据类型和较强的数据处理能力。C 语言不但具有整型、实型、双精度型，还具有结构、联合等构造类型，并为用户提供了自定义数据类型。此外，C 语言还具有预处理能力，能够对字符串或特定参数进行宏定义。

（5）C 语言具有较强的移植性。C 语言程序本身并不依赖于计算机的硬件系统，只要在不同种类的计算机上配置 C 语言编译系统，即可达到程序移植的目的。

（6）C 语言不但具有高级语言的特性，还具有汇编语言的特点。C 语言既有高级语言面向用户、容易记忆、便于阅读和书写的优点；又有面向硬件和系统，可以直接访问硬件的功能。

（7）C 语言具有较好的通用性。它既可用于编写操作系统、编译程序等系统软件，也可用于编写各种应用软件。

1.2 C语言程序

1.2.1 几个典型的C程序

C 语言的源程序由一个或多个函数组成，每个函数完成一种指定的操作，所以有人又把 C 语言称为函数式语言。下面，通过 3 个简单的例子来了解 C 程序的基本结构。

【例 1.1】运行时在屏幕上显示信息“Hello!”。

```
main()
{
  printf("Hello! \n");
}
```

运行结果：

```
Hello!
```

程序说明：

（1）C 程序由一系列函数组成，这些函数中必须有且只能有一个名为 main 的函数，这个函数称为主函数，整个程序从主函数开始执行。在【例 1.1】中，只有一个主函数而无其他函数。

（2）程序第一行中的 main 是主函数的函数名，main 后面的一对小括号是函数定义的标志，

不能省略。

（3）程序第三行的 printf 是 C 语言的格式输出函数，在本程序中，printf 函数的作用是输出括号内双引号之间的字符串，其中“\n”代表换行符。第三行末尾的分号，则是 C 语句结束的标志。

（4）程序第二行和第四行是一对大括弧，在这里表示函数体的开始和结束。一个函数中要执行的语句都写在函数体中。

【例 1.2】求两个数之和。

```
main()
{
 int a,b,sum;      /* 定义三个变量 */
 a=10; b=26;       /* 分别给变量 a 和 b 赋值 */
 sum=a+b;          /* 求 a 和 b 的和，并把结果放入变量 sum 中 */
 printf("sum=%d\n",sum);                    /* 输出变量 sum 的值 */
}
```

运行结果：

```
sum=36
```

程序说明：

（1）这个程序由一个主函数组成，其中，第三行的 int 表示定义变量类型为整型，该行定义了 a、b、sum 3 个整型变量。

（2）程序第四行和第五行中的语句均为赋值语句，“=”为赋值运算符，其作用是将其右边的常量或表达式值赋值给左边的变量。

（3）第六行中的“%d”是输入/输出函数中的格式字符串，在此表示以十进制整数的形式输出变量 sum 的值。程序的运行结果中，“%d”的位置被 sum 的值取代。

（4）程序中多次出现的“/*”和“*/”是一对注释符，注释的内容写在这对注释符之间。注释内容对程序的编译和运行不起任何作用，其目的是为了提高程序的可读性。在必要的地方给程序加上注释是一个好习惯，这使得程序易于理解，而对程序的理解是进一步修改和调试程序的基础。

【例 1.3】求一个数的平方。

```
main()
{
 int a,p;              /* 定义两个整型变量 */
 scanf("%d",&a);   /* 输入一个整数到变量 a 中 */
 p=f(a);               /* 调用 f 函数求 a 的平方，并把函数的返回值赋值给变量 p */
 printf("%d",p);    /* 输出变量 p 的值 */
}
f(n)              /* 定义 f 函数，n 为形式参数 */
int n;            /* 定义形式参数的类型 */
{
 int t;
 t=n*n;           /* 求 n 的平方，并把结果赋值给变量 t */
 return t;        /* 返回变量 t 的值 */
}
```

运行结果：

```
5 ↙
```

```
25
```

程序说明：

（1）这个程序的功能是，输入一个整数，输出该数的平方值。程序由两个函数组成，一个是 main 函数，一个是 f 函数。这两个函数的定义是相互独立的。

（2）f 函数的功能是求 n 的平方，并返回 n 的平方值。main 函数的第五行调用 f 函数时，把变量 a 的值传递给形式参数 n，因此，调用 f 函数的结果是求得了 a 的平方。

1.2.2 C程序的基本结构

通过上面 3 个例子，可以把 C 程序的基本结构归结如下：

（1）C 语言程序由函数构成。函数是构成 C 程序的基本单位，即 C 程序由一个或多个函数组成，其中必须有且只能有一个名为 main 的主函数。如在前面的【例 1.1】和【例 1.2】中，均只有一个 main 函数，而在【例 1.3】中，则有 main 和 f 两个函数。

（2）每个函数的基本结构如下：

```
函数名()
{
 语句 1;
……
语句 n;
}
```

有的函数定义时，函数名后的小括号内有形式参数，如【例 1.3】中的 f 函数。{}内则是由若干语句组成的函数体，每个语句必须以分号结束。C 语言的书写格式较自由，一行内可以写多个语句（如【例 1.2】程序的第四行有两个赋值语句），一个语句很长时也可以分写在多行上。

（3）各个函数的定义是相互独立的。各函数定义的顺序无关紧要，主函数可以定义在其他函数之前，也可以定义在其他函数之后，但程序的执行总是从主函数开始。

1.2.3 C语言的基本符号与词汇

任何程序设计语言都规定了自己的一套基本符号和词汇，C 语言也不例外。

1. C语言的基本符号集

C 语言的基本符号集采用 ASCII 码字符集，包括：

（1）大小写英文字母各 26 个。

（2）10 个阿拉伯数字 0～9。

（3）其他特殊符号，包括以下运算符和操作符：

+	−	*	/	%	<
<=	>	>=	==	!=	&&
\|\|	!	&	\|	~	=
++	--	?:	<<	>>	()
[]	.	->	^	#	sizeof
+=	-=	*=	/=	%=	&=
^=	\|=	,			

2. C语言的词汇

（1）标识符。程序中用来标识变量名、函数名、数组名、数据类型名等的有效字符序列称为标识符。简单地说，标识符就是一个名字。

标识符的构成规则如下：

① 标识符只能由英文字母（A～Z，a～z）、数字（0～9）和下划线（_）三类符号组成，但第一字符必须是字母或下划线。例如，下面的标识符是合法的：

sum，Sum，n2，_average，a_3，student_2_name

下面是不合法的标识符：

num-1，a#3，2student，!sum_2，number.3

② 大写字母与小写字母含义不同。如 sum、Sum、SUM 表示 3 个完全不同的标识符。

③ 一般的 C 编译系统只取标识符的前 8 个字符为有效字符，而 Turbo C 则取标识符的前 32 个字符为有效字符。

④ 通常，命名标识符时应该做到“常用取简，专用取繁”。一般情况下，标识符的长度在 8 个字符以内就可以了。

（2）关键字。关键字又称为保留字，是 C 语言编译系统所固有的、具有专门意义的标识符。C 语言的关键字有 32 个，一般用作 C 语言的数据类型名或语句名，如表 1-1 所示。

表 1-1　C 语言关键字

描述类型定义	描述存储类型	描述数据类型	描述语句
Typedef	auto	char	break
Void	extern	double	continue
	static	float	switch
	register	int	case
		long	default
		short	if
		struct	else
		union	do
		unsigned	for
		const	while
		enum	goto
		signed	sizeof
		volatile	return

说明：

① 所有关键字的字母均采用小写。

② 关键字不能再作为用户的常量、变量、函数和类型等的名字。

1.3 C语言集成开发环境

1.3.1 Turbo C集成开发环境介绍

C 语言有许多种编译器，这些编译器之间只有很小的区别，只要学会其中的一种，对其他几种就能很快的适应。本节主要介绍在微机上较为流行的 C 语言版本——Turbo C。

Turbo C 是美国 Borland 公司的产品。它是一个集程序建立、编辑、编译、连接、运行于一体的集成开发环境，是一个易学、易用、快速高效、方便用户使用的程序开发工具。Turbo C 具有图形库和文本窗口函数库，具有查错及协处理器的仿真功能。当前在国内使用比较普遍的是 Turbo C 2.0，它是在 MS-DOS 操作系统上使用的编译系统，也可以在 Windows 操作系统下运行。本书的所有例题都是在该环境下调试通过的，后面的介绍都以此为基础。

1.3.2 Turbo C集成开发环境的使用

1. 启动Turbo C

假设 Turbo C 安装在 C 盘根目录下的 TC 文件夹中，那么在 Windows XP 下运行 Turbo C 的步骤如下：

（1）单击屏幕左下角的“开始”按钮，选择“所有程序”/“附件”/“命令提示符”，打开“命令提示符”窗口，如图 1.1 所示。

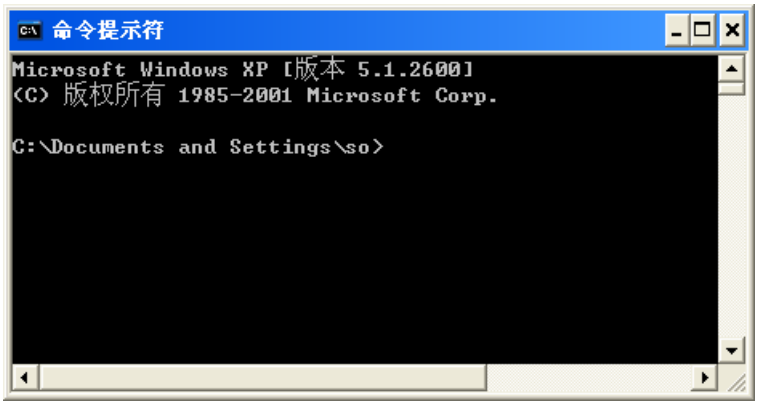


图 1.1 “命令提示符”窗口

（2）在提示符后输入“cd\”，按回车键后回到 C 盘根目录下。

（3）在提示符后依次输入以下命令即可启动 Turbo C：

```
C:\>cd tc ↙
C:\TC>tc ↙
```

Turbo C 启动后，屏幕上即显示出其开始界面，如图 1.2 所示。

2. C源程序的建立和编辑

（1）源程序的建立。在 Turbo C 中新建一个 C 源程序时，可以直接选择“Edit”（编辑）菜单，当光标停留在编辑窗口中时，就可以输入程序了。每输完一行后，按回车键将光标移到下一行的开始位置。

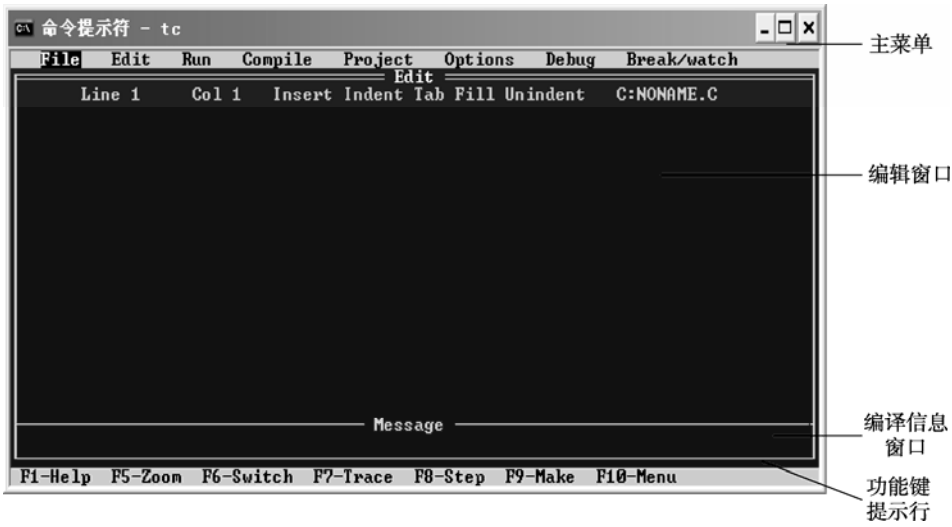


图 1.2　Turbo C 2.0 的开始界面

这里，在编辑窗口中输入【例 1.1】中的程序，如图 1.3 所示。

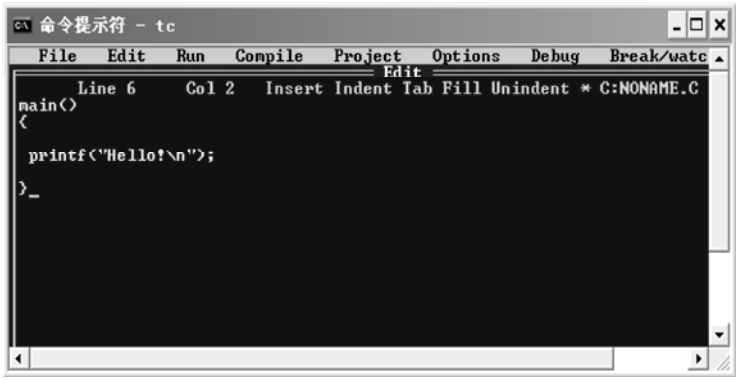


图 1.3　输入【例 1.1】中的程序

（2）源程序的编辑。在编辑窗口中，可以很方便地对源程序做修改。表 1-2 列出了 Turbo C 的常用编辑命令。

表 1-2　Turbo C 的常用编辑命令

功能键	功　能	功能键	功　能
↑	光标上移一行	Delete	删除光标处的字符
↓	光标下移一行	Backspace	删除光标处的前一个字符
←	光标左移一个字符	Ctrl-Y	删除光标所在行
→	光标右移一个字符	Ins	插入、改写状态转换开关
PgUp	光标上移一页	Ctrl-K-B	定义块首
PgDn	光标下移一页	Ctrl-K-K	定义块尾
Home	光标移到当前行首	Ctrl-K-V	移动定义的块到光标所在处
End	光标移到当前行尾	Ctrl-K-C	复制定义的块到光标所在处
Ctrl-Home	光标移到源程序首	Ctrl-K-Y	删除定义的块
Ctrl-End	光标移到源程序尾	Ctrl-K-H	取消块的定义

3. 保存C程序

选择“File/Save”（文件/保存）菜单，或直接按“F2”键，即可保存当前编辑窗口中的 C 程序，保存之前要求先给当前默认文件名为“NONAME.C”的 C 程序重命名。

4. 编译、连接和运行

用任何一种高级语言编写的源程序都不能直接运行。C 源程序在运行之前必须经过编译和连接两个步骤。编译的结果是生成一个扩展名为.OBJ 的目标代码文件，连接的结果则是生成一个扩展名为.EXE 的可执行文件。

选择“Compile”（编译）菜单中的命令可以完成编译和连接操作。更为简单直接的办法则是，选择“Run”（运行）菜单中的“Run”命令，自动完成程序的编译、连接和运行 3 个步骤。

程序运行结束后将自动回到编辑状态。若要看清运行结果，则可按以下步骤进行：

（1）按“F10”键激活主菜单。

（2）选择“Run/User Screen”（运行/用户屏幕）菜单，或直接按“Alt-F5”键，即可看到刚才的运行结果，如图 1.4 所示。

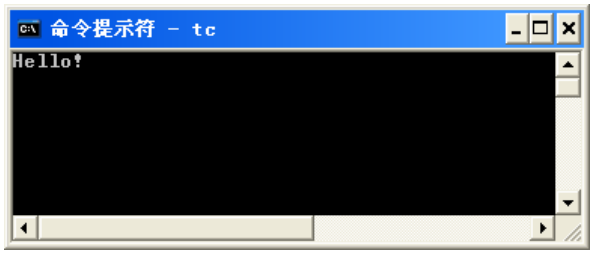


图 1.4 【例 1.1】的运行结果

（3）按任意键后，可回到编辑状态。

如果编译或连接的过程中出现错误，则出错信息将会显示在“Message”（信息）窗口中。这时，应该根据出错信息的提示，回到编辑状态修改程序。

5. 装入一个已有的C源程序

在 Turbo C 中装入一个存储设备上已有的 C 源程序的步骤如下：

（1）选择“File/Load”（文件/装入）菜单。

（2）在“Load File Name”提示框中输入要装入的路径名及文件名后按回车键，编辑窗口中即可显示出装入的源程序内容。

6. 结束当前程序，开始编辑另一个C程序

结束当前程序编辑另一个程序的步骤如下：

（1）选择“File/New”（文件/新建）菜单。若当前程序没有保存，则会显示出询问是否保存当前程序的信息，如图 1.5 所示。

（2）按“Y”键保存程序后，编辑窗口将被清空，这时即可开始建立一个新的 C 源程序。

7. 退出Turbo C

选择“File/Quit”（文件/退出）菜单，或是直接按“Alt-X”键，即可退出 Turbo C。

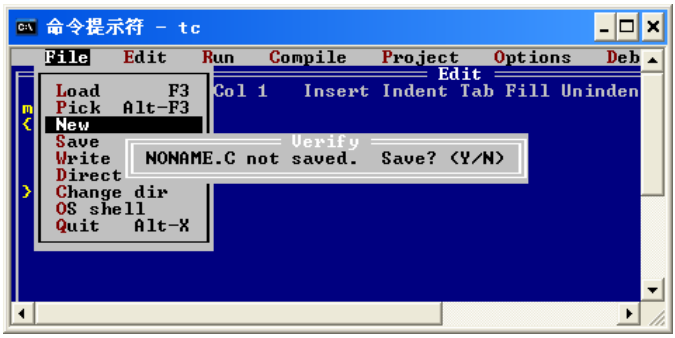


图 1.5　询问是否保存当前程序



习题一

1. 填空题

（1）C 语言程序由_____组成，其中必须有且只能有一个名为_____的函数。C 程序的执行从_____函数开始。

（2）每个 C 语句必须以_____号结束。

（3）标识符只能由_____、_____和_____三类符号构成，而且标识符的第一个字符必须是_____或_____。

（4）关键字是指_____。

（5）C 程序中，注释的内容应放在_____和_____符号之间。

2. 选择题

（1）下面合法的 C 语言标识符是_____。

A. 3ab　　B. AB.2　　C. a_3　　D. #abc

（2）C 语言中主函数的个数是_____。

A. 1 个　　B. 2 个　　C. 3 个　　D. 任意个

（3）以下有关注释的描述中，错误的是_____。

A. 注释可以出现在程序中的任何位置。

B. 程序编译时，不对注释做任何处理。

C. 程序编译时，要对注释做出处理。

D. 注释的作用是提示或解释程序的含义，帮助提高程序的可读性。

（4）C 程序中，main 函数的位置_____。

A. 必须放在所有函数定义之前。

B. 必须放在所有函数定义之后。

C. 必须放在它所调用的函数之前。

D. 可以任意。

3. 指出并改正下面程序中的错误

（1）main

```
{
printf("Welcome!");
}
```

（2）main()

```
{
 int a;
 a=5;
 printf("a=%d",a);
```

（3）main()

```
{
 int a,b
 a=1,b=2
 printf("%d",a+b)
}
```

4. 分析下列程序，写出运行结果

（1）main()

```
{
 int a,b,c;
 a=2;b=15;
 c=a*b;
 printf("c=%d",c);
}
```

（2）main()

```
{
 printf("Hello!");
 pt();
}
pt()
{
 printf("*****************");
}
```

5. 编程题

（1）编写一个程序，打印出下面的信息：

```
**********************
        Welcome!
**********************
```

（2）编写一个程序，输入变量 a 和 b 的值，输出表达式 a*b+10 的值。



上机实习指导

一、学习目标

本章是学习 C 语言的入门篇，重点介绍了 C 程序的组成结构和基本的上机操作步骤。通过本章的学习，应建立对 C 语言程序的初步认识，并能在 Turbo C 环境中建立和运行简单的 C 程序。具体要求掌握以下内容：

1. 掌握 C 程序的组成结构和 C 程序的书写格式。

2. 通过上机实习熟悉 Turbo C 环境的操作界面，并初步了解 Turbo C 系统的功能。

3. 掌握基本的上机操作步骤，包括：

（1）启动 Turbo C 系统；

（2）建立 C 源程序；

（3）运行 C 程序；

（4）修改 C 程序；

（5）保存 C 程序；

（6）装入一个已存在的 C 程序；

（7）退出 Turbo C 系统。

二、应注意的问题

1. 在程序中使用汉字

编程时有时需要通过程序输入/输出各种中文信息。由于 Turbo C 是在 DOS 环境下运行的，因此，如果需要在 Turbo C 中编辑和运行一个包含了中文信息的 C 程序，就必须在启动 Turbo C 之前运行一个中文 DOS 系统。

目前，使用较普遍的 Windows 操作系统版本是 Windows XP，而在 UCDOS、CCDOS 等多种中文 DOS 系统版本中，与 Windows XP 具有较高兼容性的则是 CCDOS97。键入以下命令即可在运行 CCDOS97 之后启动 Turbo C：

```
C:\>cd ccdos97
C:\CCDOS97>ccdos
C:\CCDOS97>cd\tc
C:\TC>tc
```

2. 同一时刻编辑窗口中只能有一个源程序

当需要建立和运行多个 C 程序时，初学者经常会出现下列错误：一个程序练习结束以后，在该程序后面直接输入另一个程序，这样是达不到目的的。千万注意，同一时刻编辑窗口中只能有一个源程序。一个程序运行结束后，如果想建立并运行另一个程序，则可以选择“File”菜单中的“Load”命令，在弹出的“Load File Name”提示框中输入新的文件名即可，或是选择“File”菜单中的“New”命令，直接开始编辑另一个新程序。

3. 对程序错误的理解

程序错误可以分为两种：编译错误和运行错误。C 语言编译器在编译阶段发现的错误称为编译错误，表现为程序中的语法错误；C 程序通过编译检查后，还需要在实际运行中检查是否有错误，称为运行错误。

编译错误可以通过静态语法检查发现，如声明变量的语法格式不正确等。运行错误不能在静态检查阶段发现，只能在程序运行中才能发现，如 0 做除数等。

一般来说，编译错误出现得更多一些，也更容易排除。运行错误出现得比较少，但比较隐蔽，难以发现。在编写 C 程序的过程中，经常会出现一些错误。对于初学者来说，这些错误似乎是很难逾越的，很难发现。但是只要从一开始就养成良好的编程习惯，严格按照语法要求，这些错误是可以克服的。



上机实习　Turbo C的基本操作

一、目的要求

1. 掌握 C 程序的基本结构。

2. 熟悉 Turbo C 系统的操作界面。
3. 能熟练地启动 Turbo C 和退出 Turbo C。
4. 掌握在 Turbo C 中建立、运行、修改、保存和装入程序的方法。
5. 掌握插入/删除字符和插入/删除行等基本的编辑操作。

二、上机内容

下面是 3 个从最简单到稍复杂的 C 程序，仔细阅读程序并在 Turbo C 中建立和运行程序，以熟悉 C 程序的基本结构和 Turbo C 的基本操作流程。

1. main()

```
{
 printf(" Welcome！ ");
}
```

（1）在编辑窗口录入该程序后，选择 Run 菜单中的 Run 命令完成程序的编译、连接和运行，最后用 Run 菜单中的 User Screen 命令重现屏幕上的运行结果。

（2）在函数体中插入一行语句，使上面的程序变成：

```
main()
{
 clrscr();          /*  清屏  */
 printf(" Welcome！ ");
}
```

再次运行程序并仔细观察运行结果，这次是先清屏，再从屏幕的左上角开始显示“Welcome！”。

2. main()

```
{
 int a,b,sum;  /*  定义 a、b、sum 3 个变量为 int 类型  */
 a=1;
 b=2;
 sum=a+b;    /*  将 a 与 b 之和赋值给 sum  */
 clrscr();
 printf("sum=%d",sum);
}
```

（1）自己先分析程序的运行结果之后再运行该程序，对比自己的判断与屏幕上的结果是否一致，如果有差异，再想想错误出在什么地方。这种做法可以逐步训练自己理解程序和分析程序的能力。

（2）删除函数体第一行中的变量定义语句“int a,b,sum;”，或将该语句注释起来，再运行程序，看看会有什么结果。

3. main()

```
{
 int a,b,c;
 scanf("%d", &a);          /*  输入一个整数到变量 a 中  */
 scanf("%d",&b);           /*  输入一个整数到变量 b 中  */
 c=max(a,b);      /* 调用 max 函数，求 a、b 中的最大值，并把结果赋值给变量 c */
printf("%d",c);  /* 输出 c 的值 */
}
max(a,b)  /*  定义 max 函数  */
```

```
int a,b;      /*   定义形式参数 a 和 b   */
{
  int t;
  if (a>b) t=a;        /*   求 a、b 中的最大值   */
else t=b;
  return t;            /*   返回变量 t 的值   */
}
```

程序中的注释是为了帮助对程序的理解，在 Turbo C 中录入程序时可以免去注释部分，以提高建立程序的效率。运行程序后仔细分析运行结果。

第 2 章　数据类型和简单程序设计



数据是语言程序处理的对象，如何编制出最好的 C 语言程序和如何在程序中描述数据，是学好 C 语言的关键。

【本章要点】

1. 有关 C 语言数据和程序设计的基本知识；
2. C 语言的 3 种基本数据类型（整型、浮点型、字符型）介绍和应用；
3. 混合运算中的类型转换。

【学习目标】

1. 了解 C 语言的数据类型、C 语句、C 程序结构的组成；
2. 弄清 3 种基本数据类型的特点；
3. 掌握 3 种基本数据类型常量的表示方法；
4. 熟练掌握 3 种基本数据类型变量的定义与使用方法；
5. 了解混合运算中的类型转换。

【课时建议】

讲授 4 课时，上机 2 课时。

2.1　基本知识

2.1.1　C语言的数据类型

任何语言程序都要对数据进行描述和处理。C 语言程序对数据的处理和操作由 C 语句来实现；而对数据的描述，主要表现在对数据类型的定义，数据属于哪一种类型要靠称为类型名的标识符加以说明。例如：整数型用 int 说明、字符型用 char 说明。

C 语言的数据类型可分为基本数据类型和导出数据类型：基本数据类型包括整数型（int）、浮点型（float）和字符型（char）3 种；导出数据类型有数组、指针、结构、联合与枚举等。在本章，主要介绍基本数据类型和使用它们的简单程序设计；数组和指针将在第 7 章和第 10 章中讲述，其他较复杂的数据类型应用请参看相关书籍或参考手册。

C 语言还有一种特殊数据类型，称为空类型（void），主要用于函数说明（详见第 7 章）。

2.1.2　常量与变量的使用

C 语言中的数据有常量和变量之分：在程序运行过程中，其值不能被改变的量称为常量，相反，在程序运行中其值可以改变的量称为变量。

1. 常量与符号常量

（1）常量。常量即常数，也有类型的区分，其类型从数的字面形式就能判别出来。如：15

和 20 是整数，即整型常量；1.5 和 2.0 是实型数，即实型常量，而'a'、'b'是字符常量（详见本章 2.4 节介绍）。

（2）符号常量。符号常量是人们为了方便和直观而在程序中自定义的一种常量。例如：

```
#define  PI  3.1416
```

这是用预处理命令定义了 PI 为符号常量，代表实型常数 3.1416。在程序中凡是使用常数 3.1416 的地方均可用 PI 来代替，既方便又直观。出现在运算式中的 PI 值是固定不变的，可多次使用，但不能再被赋值（它不同于变量，只有重新定义才能改变其值，详情请看第 9 章编译预处理宏定义一节）。

2. 变量的使用

变量的变量名和变量值是不同的概念。变量名是变量的名字，一旦被定义，便在内存中占有一定的存储单元；变量值是存放在该变量存储单元中的值，会随着给变量重新赋值而改变。变量必须先定义后使用，在程序中使用没有定义的变量，编译中会出现错误信息。

定义变量名必须符合标识符的构成规则：

（1）要用合法的字符，如字母、数字和下画线，但第一个字符不能是数字；

（2）字母大小写是有区分的，Student 和 student 不是同一个变量名；

（3）其长度一般为 8 个字符（Turbo C 允许变量名长达 32 个字符）；

（4）给变量起名最好要有意义，这不但便于记忆还能增加程序的可读性。

下面是合法的变量名：

score、Sum、_ZHJ、a、x、name1　…　等。

下面是非法的变量名：

2sum、#abc、dir:、s−1、−month　…　等。

2.1.3　C语句

C 语句是 C 程序的最基本成分，用它可以描述程序的流程控制，对数据进行处理。它与数据定义部分组成函数，若干函数和编译预处理命令（详见第 9 章）组成 C 源程序文件。

C 语句可分为单个语句、复合语句和空语句，不管什么语句都必须由分号“；”结尾。只有一个分号的语句叫空语句；用{ }括起来的多个语句构成复合语句，例如：{a=12; b=15;}是一个合法的复合语句。

从形式上分 C 语句主要有以下几种。

（1）函数调用语句：一个函数调用加上一个分号就构成一个函数调用语句。

例如：

```
printf("这是一个函数调用语句");
```

（2）表达式语句：一个表达式加上一个分号就构成一个表达式语句。

例如：　a=b+3;

请注意，若无分号只是一个赋值表达式而不是语句，加上分号便构成了一个赋值语句，它将 b+3 的值赋给变量 a。

（3）条件语句：该语句控制程序分支，满足条件一个走向，不满足条件另一个走向。例如：if() else 语句和多分支选择语句 switch 等。

（4）循环语句：该语句控制一部分语句重复执行。例如：while 语句、do−while 语句、for 语句等。

（5）其他语句。例如：无条件转向语句 goto、中止执行语句 break、返回语句 return 等。

各语句的详细情况将在以后各章中加以介绍。

2.1.4 结构化程序设计方法

随着高级语言的蓬勃兴起，人们开始注重研究程序设计中的一些最基本的问题。例如，程序的基本组成部分是什么？应该用什么样的方法来设计程序？如何保证程序设计正确等。程序的组成可以简单地描述为：程序=数据结构+算法。数据结构说的是数据类型及其组织形式，算法是对数据进行操作的步骤，从普遍意义上讲，算法就是处理问题的方法和步骤。一个好的程序设计就是要设计出好的数据结构和巧妙的算法。1969 年，E.W.Dijkstra 首先提出了结构化程序设计的概念，它强调了从程序结构和风格上来研究程序设计。经过人们多年的探索和实践，结构化程序设计方法得到了广泛的确认和推广。

结构化程序设计方法，采用自顶向下逐步求精的设计原则和单入口单出口的结构来构造程序。自顶向下逐步求精，就是先把一个复杂的问题逐步分解和细化成许多小的、容易解决的模块，然后再把模块的功能逐步分解细化为一系列具体的处理步骤和将其实现的程序语句。用好结构构造程序，是指任何程序均应由顺序、选择和循环 3 种基本结构组成；任何程序模块不论大小，应只有一个入口和一个出口，没有死语句、没有死循环、不允许随意跳转。

采用结构化程序设计方法设计出的程序具有结构清晰、层次分明、可靠性强等优点，可以提高程序的可读性、可维护性及生产效率。

C 语言支持结构化程序设计方法，所以 C 程序结构亦由 3 种基本程序结构组成。人们在进行程序设计时，为了形象直观、容易理解，常用图形工具来描述程序结构和流程控制。最常用的图形工具有传统的流程图和 N-S 图（亦称盒图）两种，下面给出的 3 种基本程序结构就是用这两种图形工具作描述的。

（1）顺序结构。如图 2-1 所示，先执行 A 后执行 B，两者是顺序执行的关系。图中（a）为传统流程图，（b）为 N-S 图（下同）。

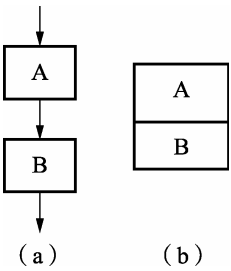


图 2-1 顺序结构

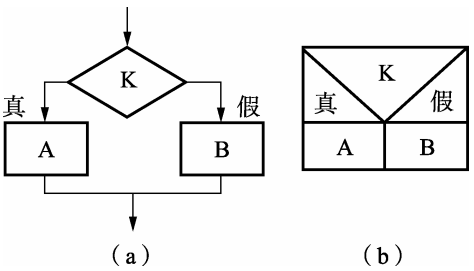


图 2-2 选择结构

（2）选择结构。如图 2-2 所示，如果条件 K 成立则执行 A，否则执行 B。

（3）循环结构。图 2-3 是 while 型循环结构，当 K 条件成立时，反复执行 A，当 K 条件不满足时结束循环。图 2-4 是直到型循环结构，先执行 A 再判断 K 条件是否满足，不满足反复执行 A，直到满足时退出循环。

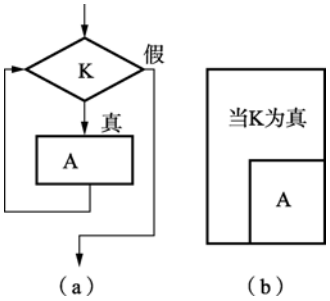


图 2-3　当型循环结构

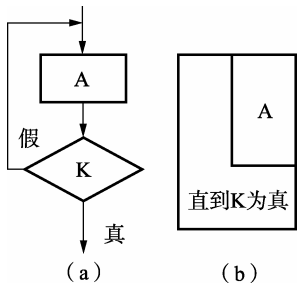


图 2-4　直到型循环结构

2.2　整型数据

2.2.1　整型常量

整型常量即整常数。C 语言中的整常数有 3 种表示形式：

（1）十进制整数　如 234、−455 等。

（2）八进制整数　如 0123、−0234 等，八进制整数以前导 0（零）开头，0123 表示八进制数 123，等于十进制数 83，即 $(123)_8=1\times8^2+2\times8^1+3\times8^0=(83)_{10}$。

（3）十六进制整数　如 0x12、−0x123 等，十六进制数以前导 0x 开头，0x12 表示十六进制数 12，等于十进制数 18，即 $(12)_{10}=1\times16^1+2\times16^0=(18)_{10}$。

为了说明整型常量的 3 种表示方法及其相互关系，请看下面例题。

【例 2.1】

```
main()
{
int x=123,y=0123,z=0x123;
pintf("%d %d %d/n",x,y,z);
printf("%o %o %o/n",x,y,z);
printf("%x %x %x/n",x,y,z);
}
```

运行结果为：

```
123   83   291
173   123   443
7b   53   123
```

例中%d、%o、%x 分别是 printf()函数的输出十进制，八进制，十六进制整型数的格式转换控制符。显示时，它们由后面自变量的值进行替换，其关系如图 2-5 所示，更详细的说明，请见第 4 章 4.2 节。

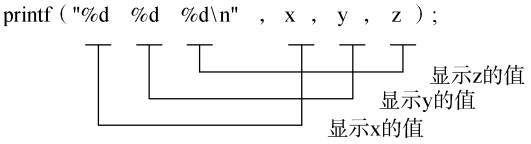


图 2-5　值的替换关系

2.2.2　整型变量及分类

整型变量通常可分为四类：一般整型（int）、短整型（short）、长整型（long）、无符号型。

其中，无符号型又有无符号整型 (unsigned int)、 无符号短整型（unsigned short）和无符号长整型（unsigned long）之分。

变量在内存中都占据着一定的存储长度，随存储长度不同，所能表示的数值范围也不同，运行在 PC 上的整型变量的字节长度和取值范围可见表 2-1。

表 2-1 整数类型的字节长度和取值范围

数据类型	字节长度	取值范围	
int	2	−32 768～32 767	即-2^{15}～$(2^{15}-1)$
short	2	−32 768～32 767	即-2^{15}～$(2^{15}-1)$
long	4	−2 147 483 648～2 147 483 647	即-2^{31}～$(2^{31}-1)$
unsigned int	2	0～65 535	即 0～$(2^{16}-1)$
unsigned short	2	0～65 535	即 0～$(2^{16}-1)$
unsigned long	4	0～4 294 967 295	即 0～$(2^{32}-1)$

下面举例说明在 PC 上，整型数据各类型所能表示的范围。

【例 2.2】

```
main()
{
int a,b;                /*定义 a,b 为整型变量*/
long c,d;               /*定义 c,d 为长整型变量*/
unsigned e,f;           /*定义 e,f 为无符号整型变量*/
a=32767;b=1;            /*为 a,b 赋初值*/
c=2147483647;d=1;       /*为 c,d 赋初值*/
e=65535;f=1;            /*为 e,f 赋初值*/
printf("int:%d,%d\n",a,a+b);
printf("olng:%ld,%ld\n",c,c+d);
printf("unsigned:%u,%u\n",e,e+f);
}
```

运行结果：

```
int:32767, −32768
long:2147483647, −2147483648
unsigned:65535,0
```

本例中，定义 a 为整型变量、c 为长整型变量、e 为无符号整型变量，在给这 3 个变量赋值时，考虑了其能容纳数的范围。若将 40000 赋给 a，将 70000 赋给 e，将 3000000000 赋给 c 则会产生溢出错误。

在 printf()函数中，使用的%ld 和%u，分别是输出长整型数据和无符号整型数据的格式转换控制符。除此之外，与【例 2.1】中不同的是双引号括起来的部分还有其他字符，输出时照原样显示，如图 2-6 所示。有时，在一个整型常量后面也加上一个字母 l 或 L，把它作为 long int 型常量，如 123l、456L 等。

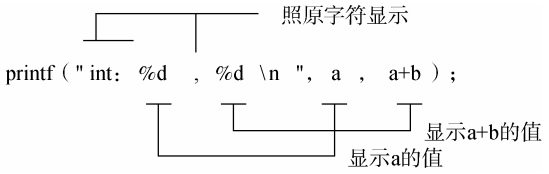


图 2-6

2.2.3　整数运算的程序设计

C 语言程序在第 1 章中已做过介绍，它与其他高级语言相比，既有相同之处，又有自己的特点，主要表现为 C 程序是由函数构成的。一个函数要有说明部分和执行部分，执行部分由各种语句组成，完成对数据的加工。在这一节中，我们练习整数运算的简单程序设计。

【例 2.3】整数型变量的四则运算。

```
main()
{
int a,b,sum,min,tim,sep;   /*整数型变量说明*/
a=7;b=2;                   /*给变量赋值*/
sum=a+b;                   /*加运算*/
min=a-b;                   /*减运算*/
tim=a*b;                   /*乘运算*/
sep=a/b;                   /*求商*/
printf("%d+%d=%d\n",a,b,sum);
printf("%d-%d=%d\n",a,b,min);
printf("%d*%d=%d\n",a,b,tim);
printf("%d/%d=%d\n",a,b,sep);
}
```

运行结果：

```
7+2=9
7-2=5
7*2=14
7/2=3
```

程序中，在第 3 行进行变量类型说明，将变量 a，b，sum，min，tim，sep 均定义成整型变量。第 4 行是赋值语句，将数值 7 赋予变量 a，数值 2 赋予变量 b，其中"="为赋值运算符。第 5～8 行计算变量 a 与 b 的和，差，积，商，并赋值给左边的变量。当两个整型变量求商时，其商为整型数值，如 7/2，商为 3 而不是 3.5。第 9～12 行显示结果，其输出格式转换请见图 2-7。

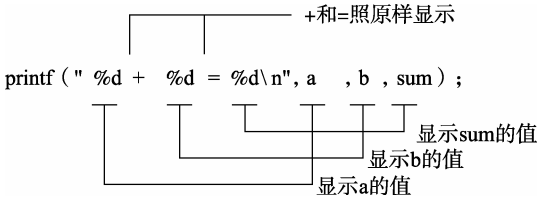


图 2-7

【例 2.4】直接输出数值计算结果。

```
main()
{
printf("%d*%d=%d\n",35,25,35*25);
}
```

运行结果：

```
35*25=875
```

程序中在 printf 语句的输出格式"%d*%d=%d"后，当写入数值 35，25，35*25 时，第一个%d 输出数值 35，第二个%d 输出数值 25，最后一个%d 直接输出数值计算 35*25 的结果，即 875。

这种简单运算，在输出语句中就完成了，免去了变量说明、运算赋值等语句，这也体现了 C 语言的灵活性。

2.3 浮点型数据

2.3.1 浮点常数

浮点型是用来表示实数的数据类型，因而也叫作实型数据。浮点常数在 C 语言中有两种表示形式。

（1）一般形式：它由数字和小数点组成，书写时，小数点不可默认，如 1.23、123、123、0.123、123.0、0.0 等都是合法的。

（2）指数形式：它靠字母 e 或 E 连接两边的数字组成，如 2.13e−27（或 2.13E−27）代表 $2.13*10^{-27}$。e 的两边必须有数，且 e 后的指数部分必须是整型数。如 4.5e、e+3、e−6、0.3e4.5 都是非法的。

注意

C 语言编译系统在用指数形式输出实型数时，是按一定的规范输出的。它是将 e 或 E 前边带有小数点的数，只取小数点左边一位非 0 的数字。例如：2132.11 可以表示为 0.213211e4、2.13211e3、21.3211e2、213.211e1、2132.11e0、21321.1e−1 等，但系统输出时，是按 2.13211e3 的格式输出(有的系统还规定指数部分的输出宽度，如：输出 e3 为 e+003 要占 5 位)。

2.3.2 浮点型变量与分类

浮点型变量有单精度型（float）和双精度型（double）之分。 运行在 PC 上的浮点型变量的字节长度和取值范围可见表 2-2。

表 2-2 浮点类型的字节长度和取值范围

数据类型	字节长度	取值范围	有效位
float	4	1.0e−38～1.0e+38	7
double	8	1.0e−308～1.0e+308	15

一般来说，double 型比 float 型精度要高，在使用时，对浮点变量要给以说明，如：

```
float    x ;
double   y;
x=111111.111
y=111111.111
```

其中，在第 1～2 行是说明语句，将 x 定义成单精度浮点型变量，将 y 定义成双精度浮点型变量，在第 3～ 4 行将相同的浮点常数赋给了不同的变量，请注意，浮点常数不分 float 型和 double 型。 一个浮点常数可以赋给一个 float 型或 double 型变量，根据变量的类型截取浮点常数中的有效位数字，如 x 只能接收 7 位有效数字，最后两位小数将失效，而 y 就能全部接收上述 9 位数字（双精度数可以提供 15 位有效数字）。

注意

实型常量都是双精度型，如果要指定其为单精度型，应该加后缀 f，如：213.21f。

2.3.3　浮点数运算的程序设计

【例 2.5】编写将 523.4562 赋给变量 a，26.2453 赋给变量 b，求其和与商的程序。

```
main()
{
float a=523.4562, b=26.2453,sum,sep;
sum=a+b;
sep=a/b;
printf("%f+%f=%f\n",a,b,sum);
printf(%f/%f=%f\n",a,b,sep);
}
```

运行结果：

```
523.456177+26.245300=549.701477
523.456177/26.245300=19.944759
```

该例的第 3 行中，不但说明了 a、b、sum、sep、是单精度浮点型变量，同时还给 a 与 b 赋了值，我们把说明语句和赋值语句合并成一个语句的形式叫作对变量进行初始化。语句只对变量 a 和 b 进行了初始化。该语句等效于：

```
float sum;
float sep;
float a=523.4562;
float b=26.2453;
```

在变量定义的语句中，可以对变量的一部分赋初值。如：

```
float x,y,z=123.45;　（相当于只对 z 进行了初始化，初值为 123.45）
```

也可以对几个变量赋以同一个初值，如：

```
float x=y=z=123.45
```

在程序的 6～7 行中，使用的输出格式转换控制符为%f，f 是 float 的缩写。注意使用 f 格式符输出实数时，以小数形式输出，在不指定输出宽度限制时，系统自动将整数部分全部输出，小数部分输出 6 位。因而 a、b 变量尽管在初始化时只有 4 位小数，但输出时都补足了 6 位。千万不要以为凡是显示出来的数字都是准确的，前面已经讲到，单精度数只有 7 位数字是有效数字。至于 a 变量在赋值时为 523.4562，在输出时为 523.456177，这是由于浮点数在内存中的存储误差引起的，如果怕计算精度不够，可以使用双精度浮点数。

【例 2.6】已知圆周率为 3.14159，半径为 15.35，编写求其圆面积和圆周长的程序。

```
main()
{
float a,b,pi=3.14159,r=15.35;
a=r*r*pi;
b=2*pi*r;
printf("are=%f\n",a);
printf("cir=%f\n",b);
}
```

运行结果：

```
are=740.229370
cir=96.446815
```

2.4 字符型数据

2.4.1 字符常数

字符常数是用单引号引起来的单个字符，如'a'、'b'、'$'、'9'、' '等。但有些特殊字符不好使用这种表示方法，而用带反斜杠的扩展表示法。如前面在 printf()函数中已经碰到过的'\n'，其中的"n"不代表字母 n，与'\'合起来代表一个换行符。这种以反斜杠开头的字符又称“转义字符”，转义字符可以用来表示 ASCII 码字符表（详见附录 A）中的任何字符，但主要是用它来表示一般字符不好表现的控制字符，经常使用的这类字符如表 2-3 所示。

表 2-3 字符常数扩展表示法

字符常数	扩展表示法	意 义
LF	\n	换行
HT	\t	跳制表域
BS	\b	退格
CR	\r	回车
FF	\f	走纸
\	\\	反斜杠字符
'	\'	单引号字符
"	\"	双引号字符
ESC	\33 或\033	脱离键
BEL	\x07	响铃

这种扩展表示法看上去好像两个字符，但实际上只起一个字符的作用。如'\0'和'0'是不同的，'0'表示的是字符 0，而'\0' 表示的是字符 NULL，即 ASCII 码值为 0 的控制字符“空操作”。在 C 语言中，有些不能用符号表示的控制符，可以用'\'加上 1～3 位 8 进制数表示的 ASCII 码值来代表，如'\33'或'\033'表示 ESC，也可以用'\'加上 1～2 位 16 进制数来表示，如'\x07' 即控制字符 BEL，表示用蜂鸣器响铃报警。

【例 2.7】

```
main()
{
printf(" ab    c\tde\07\n");
printf("f\t\b\bg");
}
```

例中用 printf()函数直接输出双引号中的各个字符，请注意其中用扩展表示法表示的字符所起的作用。第一个输出语句在第 1 行第 1 列开始输出“□ab□□c”，然后遇到“\t”，它的作用是“跳制表域”，即从当前的制表域跳到下一个制表域。我们使用的 PC 一个制表域为 8 个字符位，则下一个输出位置为第 9 列。因而在第 9～10 列输出“de”，然后遇到“\07”，即控制字符“BEL”，机器响铃。之后遇到“\n”作用是“回车换行”。第二个输出语句在第 2 行第 1 列先输出一个“□f”，然后跳到第 9 列准备输出，又遇到连续两个控制符“\b”，则向后退两格在第 7 列输出字符“g”。程序运行结果如下：

```
□ab□□c□□de
□f□□□□g
```

注意

示例中“□”是为表示空格加上去的，在显示屏或打印纸上都不会出现该符号，后面的章节中有的也这样表示，不再说明。

2.4.2　字符变量及分类

一个字符变量用来存放一个字符，在内存中占一个字节。实际上，将一个字符常数赋给一个字符变量，并不是把该字符本身放到内存单元中去，而是将该字符的 ASCII 代码放到存储单元中。因此，字符型数据也可以像整型数据那样使用，它可以用来表示一些特定范围内的整数。通常也分为两类：一般字符类型（char）和无符号字符类型（unsigned　char），运行在 PC 上的字符数据的字节长度和取值范围可见表 2-4。

表 2-4　字符数据的字节长度和取值范围

数据类型	字节长度	取值范围
char	1	−128～127 的整数
unsigned char	1	0～255 之间的整数

既然 C 语言使字符型数据和整数型数据可以通用，那么字符型变量输出时就会有不同的输出形式，并且可以对字符型数据进行算术运算。请看以下例题。

【例 2.8】

```
main()
{
char   c1,c2;      /*定义字符型变量*/
c1=65,c2=66;       /*给字符型变量赋整数值*/
printf("%c   %c",c1,c2);
printf("%d   %d",c1,c2);
}
```

运行结果：

```
A   B
65   66
```

例中第 3 行是定义 C1 和 C2 为字符型变量，char 是 character 的缩写。第 4 行将整数 65、66 分别赋给字符型变量 C1 和 C2。因为 65、66 分别是大写字母 A 和 B 的 ASCII 码值，因此，相当于赋值语句 C1='A'和 C2='B'。内存单元中存放的是数值 65 和 66，当按"%d"的格式转换符输出变量 C1、C2 时，就直接将其 ASCII 码值 65、66 作为整数输出; 当按"%C" 的格式转换符输出变量 C1、C2 时，就将其 ASCII 码值 65、66 转换成相应字符"A"和"B"进行输出。因此，有上述运行结果。

【例 2.9】

```
main()
{
char c1,c2;
c1='A',c2='B';
c1=c1+32,c2=c2+32;
printf("%c,%d\n",c1,c1);
```

```
    printf("%c,%d\n",c2,c2);
    }
```

运行结果：

```
    a,97
    b,98
```

例中利用小写字母与它对应的大写字母的 ASCII 码值大 32 的规律，通过运算进行字符转换，请读者自己分析 C 语言对字符数据这种处理方式的优越性。

【例 2.10】

```
    main()
    {
    char c1,c2,c3,c4;
    c1='a',c2='\x61',c3=0x61,c4=97;
    printf("%c,%c,%c,%c\n",c1,c2,c3.c4);
    printf("%d,%d,%d,%d\n",c1,c2,c3,c4);
    }
```

运行结果：

```
    a,a,a,a
    97,97,97,97
```

该例中用字符、转义字符、十六进制数和整常数为字符变量赋值，结果是等价的。

2.4.3 字符串

在使用字符型数据时，经常遇到的不是单个字符，而是字符串。在 C 语言中，字符串常量用双引号括起来，如"BASIC"，"How are you?"等。不要把字符常量与字符串常量混淆起来。例如，我们定义 C 为字符变量，那么，C='a';是合法的赋值语句，而 c="a"是非法的。这是因为，C 语言规定在每一个字符串的结尾处，都要加一个字符串终止符'\0'，以便系统用来判断字符串是否结束。字符串"a"实际上包含 2 个字符：'a'和'\0'（系统自动加上去的），因此，将"a"赋给一个字符变量 C 显然是不行的。

在 C 中，没有像在 BASIC 中那样有专为存储字符串的变量(如 A$,B$等)，但可以定义一个字符型数组或字符型指针变量存储字符串。

（1）字符型数组可按下列形式说明：

```
    char   str[6];
```

该语句在编译时，将留出 6 个字符的空间，但它只能存储 5 个有效的字符，即从 str[0] 存到 str[4]，str[5] 要放一个字符串的终止符'\0'。'\0'是一个空操作字符，系统本身自动加上的，输出时只作判断字符串结束的标志，并不显示。请看下面例题。

【例 2.11】

```
    main()
    {
    char msg[10];
    strcpy(msg,"Hello!");
    puts(msg);
    }
```

该程序在编译时，遇到 char msg[10]这条语句时，编译程序在内存某处划出一个连续的 10 个字节的区域，并将第一个字节的区域地址赋予 msg。其中 strcpy()是复制函数，它将"Hello!"字符串，一个字符一个字符地复制到 msg 所指的内存区域，存储完'!'后，系统自动加上字符串

终止符'\0'（见图 2-8）。最后由 puts(msg)语句输出整个字符串，在屏幕上显示出来，但终止符'\0'为空操作，并不显示。

msg[0]	msg[1]	msg[2]	msg[3]	msg[4]	msg[5]	msg[6]	msg[7]	msg[8]	msg[9]
H	e	l	l	o	!	\0			

图 2-8　字符串在字符型数组 msg 中存储状态

该程序运行结果为：

```
Hello!
```

（2）字符型指针变量可按下列形式定义：

```
char *a;
```

语句中符号*是指针运算符，表示 a 为指针变量，整个语句表示指针 a 所指向的数据，是字符类型。定义时使用 char *a="C program"，就能把字符串“赋值给变量 a”。在 printf()函数中，可以用%S 这一转换控制符进行字符串输出，S 表示 String 的字头。请看例题。

【例 2.12】

```
main()
{
char   *a="c program";
printf("%s\n",a);
}
```

运行结果：

```
c   program
```

有关数组和指针的详细说明，请见第 6 和第 10 章。

2.4.4　最简单的字符和字符串处理程序

【例 2.13】编写一个程序，在显示"C Language"字符串后，隔一行，再显示隔列打印的单个字符'O'、'K'、'!'。

```
main()
{
printf("C Language\n");
printf("\n");
printf("%c %c %c",'O', 'K', '! ');
}
```

运行结果：

```
C Language

O K !
```

【例 2.14】 利用变量赋值，然后显示 China 的程序。

该程序可用字符变量和字符型指针变量两种形式进行设计，利用指针变量可见【例 2.12】。利用字符变量，程序如下：

```
main()
{
char   a,b,c,d,e
a='C',b='h',c='i',d='n',e='a';
printf("%c%c%c%c%c\n",a,b,c,d,e);
```

```
}
```

运行结果：

```
China
```

2.5 类型的混合运算

在 C 语言中，字符型数据与整型数据可以通用，其他不同的数据类型可以在同一表达式中进行混合运算，运算时要进行同型转换。转换方式可以有两种： 一种是自动转换(隐式转换)，另一种是强制转换(显示转换)。

2.5.1 类型的自动转换

自动转换的规则是，按照低类型转换到高类型或赋值号的右边类型转换到赋值号左边的类型来进行的。例如：

（1）字符型（char）和短整型（short）必定要先转换成整型（int）。

（2）单精度型（float）必定要先转换成双精度型（double）。

（3）若两个操作数之一是 double 型，则另一项也转换为 double 型，运算结果为 double 型；

如果两个操作数之一为 long 型，则另一项也转换为 long 型，结果为 long 型；

如果两个操作数之一为 unsigned 型，则另一项也转换为 unsigned 型，结果为 unsigned 型。

（4）赋值号右边的类型转换为赋值号左边的类型，结果为赋值号左边的类型。当把右边的浮点型转换成整型时，去掉小数部分;把右边的双精度型转换成单精度型时，进行四舍五入处理。

【例 2.15】

```
man( )
{
char a='x';
int   b=2,c=3;
long f=32L;
float d=2.5678;
double e=5.2345;
printf("%f\n",a-c+e/d-f*b);
}
```

运行结果是：

```
55.038515
```

该例中，① e/d 运算时，将 d 转换为双精度型，结果为双精度型；② 进行 f*b 运算时，将 b 转换为长整型，结果为长整型；③ 进行 a-c 运算时，将 a 转换为整型，结果为整型；④ 进行③+①-②运算时，将③转换为双精度型，+①计算结果为双精度型，再-②运算时，将②的长整型转换为双精度型，结果为双精度型。


注意

%f 输出格式是输出实型数（单、双精度均可），以小数形式输出，结果的整数部分全部输出，小数部分默认为 6 位，但并不保证所有的数字都是有效数字。2.3 节已经讲过，如果输出单精度数提供 7 位有效数字，而本例输出为双精度数，可提供 15 位有效数字，运行结果的所有数字，都为有效数字。

【例 2.16】

```
man( )
{
char a='a',c='c';
int   i1,i=238;
unsigned int u1,u=463;
long l1,l=2147483147;
float f1,f=73.98;
double d1,d=23.76;
i1=i+a;   /*i 为整型数,a 为字符,结果为整型数*/
u1=u-i;   /*i 为整型数,u 为无符号整型,结果为无符号整型*/
f1=f-c;   /*f 为单精度型,结果为单精度型*/
d1=d*f-i;/*d 为双精度型,结果为双精度型*/
printf("i1=%d,u1=%u,f1=%f,d1=%lf\n",i1,u1,f1,d1);
i1=i-d;   /*运算结果为双精度数,再转换为整型数赋给 i1*/
f1=f-d;   /*运算结果为双精度数,再转换为单精度赋给 f1*/
d1=c-d;   /*运算结果为整型数,再转换为双精度数赋给 d1*/
printf("i1=%d,%f,d1=%lf\n",i1,fi,di);
}
```

运行结果是：

```
i1=335,u1=225,f1=-25.019997,d1=1519.764880
i1=214,f1=50.220005,d1=75.240000
```

2.5.2　类型的强制转换

类型的自动转换是系统自动进行的，不需要用户干预，但有时为了达到某种目的，还必须进行类型的强制转换，不然会出现错误的运算结果。请看下面的程序。

【例 2.17】

```
man( )
{
int a=300,b=200;
long c;
c=a*b;
printf("%ld\n",c);
}
```

运行结果：

```
−5536
```

该程序没有得出正确的结果 60000。这是因为在 a*b 运算时，结果已超出整型数所能表示的范围（−32768～32767），因而产生了溢出。尽管设计程序时考虑到了用长整型变量 c 来存储运算结果，但由于在向 c 赋值之前就已产生了溢出，因而存储在 c 变量中的数据仍是错误的。要解决这个问题，应将程序的第 5 行改为：

```
c=(long)a*b;
```

这样，在取出变量 a 时，a 的值被强制转换成长整型，然后与 b 进行运算，b 自动进行类型转换成为长整型。运算结果也为长整型，再赋值给长整型变量 c，结果能输出正确的结果 60000，就不会再出现错误了。

像这样用圆括号把要换成的数据类型括起来，并放在要转换的变量前面，就能把它转换成

为()内的数据类型，我们叫作强制类型转换，请看下面例题。

【例 2.18】

```
man(  )
{
int   a1,a2;
float   b,c;
b=38.425,c=12.0;
a1=(int)(b+c);        /*将(b+c)变换成整型数*/
a2=(int)b%(int)c;   /*将 b 与 c 变换成整型数后求余*/
printf("%d\n",a1);
printf("%d\n",a2);
}
```

运行结果是：

```
50
2
```

例中第 7 行的%是求余运算符，该运算符要求它的两侧数据均为整型数，因而 b 和 c 都要强制转换成整型数后，再进行求余运算。另外，数组的下标也必须是整型数，当用非整型变量做下标时，一定要强制转换为整型数。

【例 2.19】设变量 a=15，b=368.2212，编写求其符点数和的 C 程序，要求对 a 采用强制转换方式参加求和运算。

```
#include <stdio.h>
man( )
{
int   a;
float   b,c;
a=15;
b=368.2212;
c=(float)a+b;        /*将 a 变换成符点数*/
printf("c=%f\n",c);
}
```

【例 2.20】 设 b=33.125，c=56.866，编写将 b+c 之和强制取整赋值给 a1，对 b、c 取整求和赋予 a2 的 C 程序。

请读者参考【例 2.18】程序的编写方法自己写出该程序。



习题二

1. 单项选择题

（1）C 语言中的变量名只能由字母、数字和下画线组成，且第一个字符_____。

A. 必须是字母　　B. 必须是下画线

C. 必须是字母或下画线　　D. 可以是字母、数字或下画线中任一种

（2）合法的常量是_____。

A. 'program'　　B. –e8　　C. 03x　　D. 0xfL

（3）C 语言中的基本数据类型所占存储空间长度的顺序是_____。

A. char<=int<=float<=double<=long

B. int<=char<=float<=long<=double

C. int<=long<=float<=double<=char

D. char<=int<=float<=long<=double

（4）下列符号串中与 123.0 相同的合法常量是_____。

A. 0123　　B. "123.0"　　C. 1.23e2　　D. 0x123

2. 填空

（1）语言中的基本数据类型包括_____、_____ 和_____3 种。

（2）整型常数有_____、_____和_____3 种表示形式; 整型变量可分为四类：_____、_____、_____、_____。其中 _____又可分为_____、_____和_____3 种。

（3）有 10 个用指数形式表示的浮点常数，如：

① 4.5E　② 0.3E4.5　③ 2E–2　④ +32E+2　⑤ E–6

⑥ 2.E–3　⑦ .E+3　⑧ 1E2　⑨ 4E　⑩ 0.0E10

请指出合法的有_____。

（4）在 C 中，没有专为存储字符串的变量，但可以用_____ 或_____存储字符串。其定义格式为_____和_____。

（5）数据类型混合运算时，要进行同型转换， 转换方式分为_____和_____两种。

3. 写出下列程序运行结果

（1）main()

```
{
 int   a,b,c,d;
 a=215,b=9;
 c=a/b;
 d=a%b
 printf("%d/%d=%d ••• %d\n",a,b,c,d);
}
```

（2）main()

```
{
 char   a='a',b='b';
 printf("%d\t\b%c",a,b);
}
```

（3）main()

```
{
 int   a=2,b=3;
 float   c=5.0,d=2.5;
 printf("%f",(a+b)/2+c/d);
```

（4）main()

```
{
 float   a=3.14;
 printf("a=%f\n",a);
}
```

4. 编写程序

（1）设 a 为 19，b 为 22，c 为 650，编写求 a*b*c 的程序。

（2）设 b 为 35.425，c 为 52.954，编写求 b*c，将值整数化后赋给 a1，再将 c 除以 b 得的余数赋给 a2 的程序。

（3）已知半径 *r*=15cm，圆柱高 *h*=3cm，编写求圆周长、圆面积、圆柱体积的程序。



上机实习指导

一、学习目标

本章介绍了有关 C 语言数据和程序设计的一些基本知识，重点讲述了 C 语言的 3 种基本数据类型，即整型、浮点型和字符型，以及这 3 种数据类型的简单程序设计。通过本章学习应达到以下要求：

了解 3 种基本数据类型的特点；

掌握 3 种基本数据类型常量的表示方法；

熟练掌握 3 种基本数据类型变量的定义与使用方法；

了解混合运算中的类型转换。

二、应注意的问题

1. 程序中的变量必须是“先定义，后使用。”

一个变量是一个存储单元，它能存储相同类型的不同数据，给变量定义就是要根据存储数据的需要指定变量的类型。如：想用变量存储单个字符，就定义 char 型，想存储 30000 以内的正整数，就定义 int 型，若想存储小数，则应定义为 float 型。

2. 定义变量时易出现的语法错误。

上机时，有语法错误的程序不能通过编译，系统在 Massage 窗口中给出出错信息，并在每条出错信息前有一个用来报告错误所在行的行号，以便于查找和修正。定义变量时易出现的语法错误一般有两种。

（1）变量没有定义，出错信息为：Undefined symbol 'XXXX'。

例如：main()

```
{
a=2;
b=3;
printf("%d",a+b);
}
```

该程序在编译时，将在 Massage 窗口中给出第 3 行的变量 a 和第 4 行的变量 b 没有定义的该条出错信息。

（2）定义变量书写不当，出错信息为：Declaration syntax error。

当在上述程序的第 3 行加上变量定义语句，如：

```
int a b;
```

或

```
int 2a,b;
```

在编译时，系统会给出该条出错信息。前者是因为变量 a 和 b 之间用空格做间隔；后者是因为使用了一个名字不合法的变量 2a。

当将该变量定义语句写成如下形式：

```
int a;b;
```

在编译时，系统给出的出错信息为：

```
Undefined symbol 'b' in function main
```

请自己分析这是为什么。



上机实习　基本数据类型的简单程序设计

一、目的要求

进一步熟悉 Turbo C 集成环境下程序的建立、修改和运行；

熟悉定义 3 种基本数据类型变量的方法；

初步了解简单程序设计及 Printf()函数的使用方法；

练习修改变量定义的语法错误和程序调试。

二、上机内容

1. 语法错误的检查与修改

下面的程序都含有语法错误。在 Turbo C 集成环境中建立源程序，并使用 Run 命令后，注意观察出错信息，然后分析原因、修正错误、重新运行。

当有出错信息在 Massage 窗口出现时，可以按 F6 键在 Massage 窗口与 Edit 窗口之间切换。

（1）main()

```
{
   int x1,x2;
   x1=5;
   x2=10;
   y=x1*x2;
   printf("y=%d",y);
}
```

（2）main()

```
{
   int a,b,c;
   a=2000;
   b=3000;
   c=a+b;
   printf("c=%d",c);
}
```

（3）main()

```
{
   char ch;
   ch=A;
   printf("%c",ch);
}
```

（4）main()

```
{
   float m#1;
   m#1=2.36;
   printf("%f",m#1/2);
}
```

2. 分析程序运行结果

上机前先写出下面程序的运行结果，然后上机验证自己分析的是否正确。

（1）main()

```
{
   int r;
   float s;
   clrscr();
   r=2;
   s=3.14159*r*r;
   printf("r=%d\n",r);
   printf("s=%f",s);
}
```

程序中的第 5 行用到了 C 语言提供的清屏函数 clrscr()，该函数的调用应放在输出语句之前，它可以使运行结果在清屏后再输出，显得整洁清晰。

（2）main()

```
{
   char c1,c2;
   clrscr();
   c1='a';
   c2='b';
   printf("字母 a 的 ASCII 码为：%d\n 字母 b 的 ASCII 码为：%d",c1,c2);
}
```

（3）main()

```
{
   char ch;
   clrscr();
   ch='\01';
   printf("%c\n",ch);
   ch='\05';
   printf("%c",ch);
}
```

提示：程序中用到的'\01'和'\05'分别表示 ASCII 码为 1 和 5 的字符，前者是一个笑面符号，后者是一个梅花符号。通过这种形式，可以输出不能用键盘符号表示的字符。上机时，可以将'\01'和'\05'分别改为'\03'和'\04'，看看会有什么运行结果。

3. 设计编写一个简单的程序

已知某学生的三门功课的分数为：100、80 和 75，求该学生的平均成绩是多少？

第 3 章　表达式与运算符



运算符和表达式是 C 语言程序设计中极为重要的内容，C 语言提供了丰富的运算功能和多种表达式形式。

【本章要点】

1. 表达式与运算符概述；
2. 算术运算符、算术表达式的应用；
3. 赋值运算符、自增自减运算符等其他运算符的应用。

【学习目标】

1. 了解运算符的意义、作用和使用方法；
2. 掌握常用运算符的优先级和结合性；
3. 能够灵活使用运算符构造表达式，并正确求取表达式的值。

【课时建议】

讲授 3 课时，上机 2 课时（利用机动课时）。

3.1　概述

3.1.1　表达式

1. 什么是表达式

在数学中，用运算符将常量、变量、函数连接起来的有意义的式子称为数学表达式。例如：$8+\cos(x+5.3)$，$4\times(x+y)$等都是数学表达式。

同样，在 C 语言中，把符合 C 语言规定的，用 C 语言运算符将常量、变量、函数调用连接起来的有意义的式子称为 C 语言表达式。常量、变量、函数调用本身又是最简单的表达式。C 语言的表达式分为算术表达式、赋值表达式、逗号表达式、条件表达式、关系表达式和逻辑表达式等。

2. 表达式的值

使用表达式的目的是计算表达式的值。算术表达式的值是个数值，而数值的类型，取决于参与运算的数据的类型。例如，算术表达式 300+50，其值为 350，由于参与运算的两个数据均为整型，则表达式的值也为整型。关系表达式也有值，其值是对两个量的比较结果。如果关系表达式成立，则结果为 1，代表“真”；否则，结果为 0，代表“假”。

3. 表达式与语句

表达式与语句有很密切的关系，在表达式后面加上分号“;”就构成一条语句。分号标志着

语句的结束。

例如:

```
++b;                                /* 自增运算表达式语句 */
x=a+b;                              /* 赋值运算表达式语句 */
printf("a=%d,b=%d",a,b);            /* 函数调用语句* /
```

必须说明，并不是所有的表达式组合成的语句都有意义，如下面的语句，可做求值运算，语法也正确，但无实际作用。

```
a+b;
```

表达式能构成语句是 C 语言的一个特色。 由于表达式是由运算符连接而成的，而 C 语言的运算符又相当丰富，所以 C 程序中的表达式也是相当丰富的。“函数调用语句”也属于表达式语句，因为函数调用也属于表达式的一种，因此，C 语言又被称为函数语言和表达式语言。

3.1.2 运算符

C 语言把除了控制语句和输入/输出以外的几乎所有的操作都作为运算符处理，C 语言的运算符种类繁多，具体分类见表 3-1。

表 3-1 C 语言运算符

名 称	运 算 符		
算术运算符	+、-、*、/、%、++、--		
关系运算符	>、>=、==、!=、<、<=		
位运算符	>>、<<、~、&、	、^	
逻辑运算符	!、&&、		
条件运算符	? :		
指针运算符	&、*		
赋值运算符	=、+=、-=、*=、/=、%=、&=、^=、	=、<<=、>>=	
逗号运算符	,		
字节运算符	sizeof		
强制转换运算符	(类型名)(表达式)		
其他	下标[]、成员(->、•)、函数()		

上面的运算符按参与运算的对象个数又可以这样分类：单目运算符、双目运算符和三目运算符。例如“++”和“!”等是单目运算符，“/”和“&&”等是双目运算符，C 语言中有且仅有条件运算符“？：”是三目运算符。

3.2 算术运算符与算术表达式

3.2.1 算术运算符

表 3-2 列出了 C 语言的所有算术运算符。

表 3-2　算术运算符

运算符	功　能	应用举例
+	加法运算符	a+b
-	减法运算符	a-b
+	取正数(单目加)运算符	+a
-	取负数(单目减)运算符	-a
*	乘法运算符	a*b
/	除法运算符	a/b
%	取模(求余)运算符	a%b
++	自增运算符	a++或++a
--	自减运算符	a--或--a

上面大多数运算都是读者较熟悉的，下面仅就除法、取模、取负运算做些说明。

1. 除法运算

当两个操作数都是整型数时，除法运算视为整除运算，运算结果将舍去小数部分，只保留整数部分。例如，对于整数运算，8/5 的结果为 1；而对于实数运算，8.0/5.0 的结果为 1.6。

2. 取模运算

取模运算又称求余运算，运算结果为一个整型数，这个数是整除运算的余数，符号与被除数符号相同。例如，8%5 的结果是 3；8%(−5)的结果是 3；(−8)%5 的结果为−3；　(−8)%(−5)的结果为−3。

3. 取负运算

取负运算是单目运算，即只有一个操作数参与运算。取负运算是将参与运算的操作数无条件取负。例如，−8，将常数 8 变为−8；又如，−(−8)，将−8 取负，结果为 8。

【例 3.1】取模运算符使用示例

```
main( )
{
printf("5%%3=%d,5%%-3=%d,-5%%3=%d,-5%%-3=%d",5%3,5%-3,-5%3,-5%-3);
}
```

运行结果为：

```
 5%3=2,5%-3=2,-5%3=-2,-5%-3=-2
```

注意

为了能显示“%”，相连的两个“%”相当于输出一个“%”，当然也可以用“\%”代替“%%”，结果是相同的。

3.2.2　算术表达式

1. 什么是算术表达式

用算术运算符和括号将常量、变量和函数调用连接起来的、符合 C 语言语法规定的式子，

称为 C 语言的算术表达式。

例如：

```
a*b/c+'a'-100+sqrt(10)
        10
        a
```

均为合法的算术表达式。

2. 算术运算符的优先级

（1）算术运算符的优先级：括号→函数调用→取负→*、/、%→+、−其中，括号的优先级最高，而+、−运算的优先级最低。例如，a−b*c 相当于 a−(b*c)。

在 C 语言的表达式中，只允许使用小括号（圆括号），不允许使用中括号和大括号。当出现多重括号时，先执行最内层括号，接着执行外一层，直到最后执行最外层括号。例如，表达式 a*((b+c)/d+e)，首先计算内层括号的 b+c，然后计算外层括号的除以 d，并加上 e，最后乘以 a。

（2）算术运算符的结合性：算术运算符的结合方向是“从左至右”，当操作数两侧的运算符的优先级相同时，则按规定的“结合方向”进行先左后右的顺序处理。例如，a+b−c 先计算 a+b，再执行减 c 的操作。

3. 算术表达式的使用说明

C 语言的运算符和表达式使用很灵活，利用这一点可以巧妙地处理许多在其他语言中难以处理的问题。但另一方面，有时也会出现一些令人容易搞混的问题。因此使用时务必要小心谨慎。如适当使用括号，将会避免出现二义性问题。

例如：

$\frac{a-b}{a+b}$ 不能写成 $a-b/a+b$，应写出 $(a-b)/(a+b)$

【例 3.2】算术表达式的应用举例

数学表达式	算术表达式
$2\pi r$	2*3.14*r
$2ab\sin 30^{o}$	2*a*b*sin(30*3.14/180)
$3x^4+4y^{n-1}$	3*pow(x,4)+4*pow(y,n−1)
$25[x^2+(x+y)*a]^3$	25*pow((x*x+(x+y)*a),3)
$\frac{3.6}{xy}$	3.6/(x*y)
$\frac{-b+\sqrt{b^2-4ac}}{2a}$	(−b+sqrt(b*b−4*a*c))/(2*a)

注意

例中所用到的 pow(x,y)与 sqrt(x)函数，是 C 语言提供的常用的数学库函数，是求 x 的 y 次方和对 x 开平方，其他数学函数请参阅附录 B。在编写程序时，若使用数学函数，应加上编译预处理命令和数学头文件：#include <math>（详细说明请见第 9 章）。

3.3　其他运算符的应用

3.3.1　赋值运算符和赋值表达式

1. 赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子，称为赋值表达式。其一般形式如下：

变量　赋值运算符　表达式

例如：a = 5+10

其中，a 为变量，“＝”为赋值运算符，5+10 为表达式。

赋值表达式的计算过程是：首先计算表达式的值(5+10 得 15)，然后将该值（15）赋给左侧的变量（a）。

2. 赋值运算

赋值运算的作用是将一个数据赋给一个变量。

例如：

```
a=10            将 10 赋给变量 a
b=20            将 20 赋给变量 b
c=a+b           计算 a+b 得 30，将 30 赋给变量 c
d=2*sqrt(16)    计算 2*sqrt(16)得 8，将 8 赋给变量 d
```

以上均为正确的赋值运算（注意：以上仅仅是赋值表达式，并不是赋值语句）。

3. 复合的赋值运算

在赋值运算符“=”之前加上其他双目运算符可构成其他复合赋值运算符。如+=，-=，*=，/=，%=，…等。

例如：

```
a+=b;          /*  等价于 a=a+b */
a*=b+3;        /*  等价于 a=a*(b+3) */
a%=b;          /*  等价于 a=a%b */
```

C 语言中采用复合运算符，一是为了简化程序，使程序精练；二是为了提高编译效率，产生质量较高的目标代码。

4. 赋值表达式的使用说明

（1）赋值号“＝”不同于数学中的等号，它没有“相等”的含义。

例如，数学中 x=y*y 成立，那么 y*y=x 也成立；但 C 语言中的 x=y*y 并不表示 x 等于 y*y，而是将 y*y 的值送入 x 中。因此赋值号两边的内容不能交换，上式不能写成 y*y=x。

在 C 语言中，常常可以看到下面这种表达式形式：

```
n=n+1
```

在数学中，n=n+1 很难理解，因为 n 不等于 n+1；但是在 C 语言中却不难理解，它表示将 n 原有的值加 1，再送回 n 中去，此时 n 的原有值被新值替换了。

（2）赋值号左边只能是变量，不允许出现常量、函数调用或表达式。

（3）赋值表达式中的“表达式”，又可以是另一个赋值表达式。

例如：

```
x=(y=3*a)
```

（4）赋值运算符的优先级低于算术运算符、关系运算符和逻辑运算符，但高于逗号运算符；赋值运算符的结合性是“从右至左”。

例如：

```
x=y=z=10
```

其运算顺序是从右至左结合，即先执行 z=10，然后再把结果赋给 y，最后把 y 的结果值 10 赋给 x。

又如：设 a 的值为 10：

```
a+=a-=a*a
```

首先计算 a−=a*a，它相当于 a=a−a*a=10−100=−90；

再计算 a+=−90，它相当于 a=a+(−90)= −90+(−90)= −180。

即，上式相当于：

```
a=a+(a=a-a*a)
```

（5）当赋值号两边的数据类型不同时，一般由系统自动进行类型转换。其原则是，赋值号右边的数据类型转换成与左边的变量相同的数据类型。

5. 赋值表达式的应用举例

【例 3.3】赋值表达式

```
a=10+(b=6)          b 的值为 6，a 的值为 16，表达式的值为 16。
a=(b=4)+(c=16)      b 的值为 4，c 的值为 16，a 的值为 20，表达式的值为 20。
a=(b=20)/(c=10)     b 的值为 20，c 的值为 10，a 的值为 2，表达式的值为 2。
```

3.3.2 自增和自减运算符

1. 自增、自减运算符的功能

自增、自减运算符的作用是使变量的值增 1 或减 1。但自增、自减运算符的前置（在变量之前）和后置（在变量之后），其含义是不同的。

例如：

```
++i,--i   表示在使用 i 之前，先使 i 的值加（减）1；
i++,i--   表示在使用 i 之后，再使 i 的值加（减）1。
```

【例 3.4】已知 i 的值为 10，则：

```
j=++i     i 的值自增为 11，j 的值为 11。
j=--i     i 的值自减为 9，j 的值为 9。
j=i++     j 的值为 10，i 的值自增为 11。
j=i--     j 的值为 10，i 的值自减为 9。
```

从【例 3.4】可以看到，对于前置运算，变量 i 本身先自增（或自减），然后再赋与变量 j；而对于后置运算，变量 i 的值先赋与变量 j，然后变量 i 本身再自增（或自减）。

自增、自减运算符只能用于整型变量，不能用于常量或表达式。例如，8++或(b+5)++是不合法的，因为 8 是常量，而表达式（b+5）的结果是数值，不是变量。

2. 自增、自减运算符的优先级与结合性

自增、自减运算符的优先级与“取正负”的优先级相同，即优先于“乘、除、取模”运算。

自增、自减运算符的结合方向是“从右至左”的“右结合性”。

例如：

```
-i++
```

如果按左结合性，相当于(−i)++，但这是不合法的，因为对表达式(−i)不能进行递增、递减运算。正确的方法应该是“右结合性”，即相当于− (i++)。

3. 自增、自减运算符的使用说明

（1）在表达式中使用自增或自减运算时，很容易出错。

【例 3.5】设 j 的原值为 2，计算以下表达式:

```
(j++)+(j++)+(j++)
(++j)+(++j)+(++j)
```

其中，第一式中是计算 2+3+4，还是计算 2+2+2 呢？第二式中是计算 3+4+5，还是 5+5+5 呢？我们可以通过下面的程序验证。

```
main()
{
 int   j=2;
 printf("j=%d,(j++)+(j++)+(j++)=%d\n",j,(j++)+(j++)+(j++));
 j=2;
 printf("j=%d,(++j)+(++j)+(++j)=%d\n",j,(++j)+(++j)+(++j));
}
```

程序的运行结果如下：

```
j=5,(j++)+(j++)+(j++)=9
j=5,(++j)+(++j)+(++j)=12
```

注意

输出的 j 值为 5，这是因为函数调用中参数按从右到左的顺序求值。

（2）书写表达式时，应注意括号的运用。

如果有以下表达式：

```
i+++j
```

是理解为(i++)+j，还是 i+(++j)呢?C 语言编译系统在处理时尽可能多地从左至右将若干个字符组成一个运算符（在处理标识符、保留字时也按同一原则进行）。因此上式应理解为(i++)+j 而不是 i+(++j)。

3.3.3　条件运算符和条件表达式

1. 条件运算符

C 语言提供了一个唯一的三目运算符(？：)，即条件运算符。三目运算符的含义是指有 3 个量参与运算。它可以组成一个条件表达式，其一般形式为：

表达式 1？表达式 2：表达式 3

2. 使用说明

（1）使用条件运算符的条件：如果判断表达式为“真”或“假”时，都只执行一个赋值运算，并且给同一个变量赋值时，可以使用条件运算符。

例如，如果 x>y 则将 x 的值赋给 a，否则将 y 的值赋给 a，便可组成条件表达式赋值语句。

如：a=x>y?x:y；相当于

```
if(x>y)
   a=x;
else
   a=y;
```

（2）条件表达式的执行过程：首先计算表达式 1，如果非 0（真）则计算表达式 2，表达式 2 的值作为条件表达式的值；如果表达式 1 的值为 0（假），则计算表达式 3，表达式 3 的值作为条件表达式的值，如图 3-1 所示。

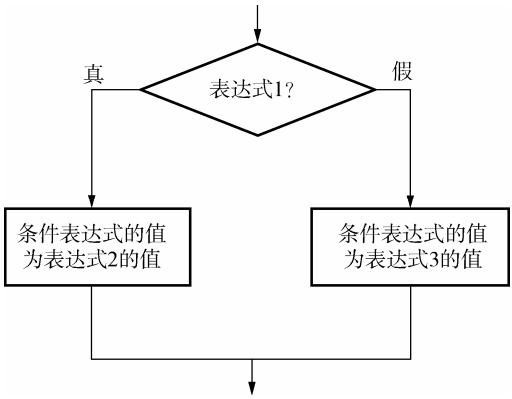


图 3-1　条件表达式执行示意图

（3）条件运算符的优先级与结合性：条件运算符优先于赋值运算符，但低于关系运算符和算术运算符。

例如：

```
x=a>0?a*10:a*(-10)
```

运算顺序相当于：

```
x=(a>0)?(a*10):(a*(-10))
```

所以括号可以省略不写。

条件运算符的结合方向是“从右至左”。

例如：

```
x>y?x:y>z?y:z
```

运算顺序相当于：

```
x>y?x:(y>z?y:z)
```

如果 x=1、y=2、z=3，则条件表达式的值为 3。

（4）表达式 1 与表达式 2、表达式 3 的类型可以不同。

例如：

```
a? 'b': 'c'
```

表达式 2、表达式 3 的类型也可以不同。

例如：

```
a>b?2:5.5
```

该表达式的值的类型取决于 a>b 的结果，如果 a>b 成立，则以表达式 2 的类型为最终类型，否则，以表达式 3 的类型为最终类型。

【例 3.6】利用条件表达式求 a、b、c 3 个数中的最大值。

```
main( )
```

```
{
 float a,b,c,max;
 scanf("%f,%f,%f",&a,&b,&c);
 max=a>b?a>c?a:c:b>c?b:c;
 printf("max=%f\n",max);
}
```

运行结果为：

```
2.34,4.56,1.87↙
max=4.560000
```

读者一定会认为程序中的语句 max=a>b?a>c?a:c:b>c?b:c;太难理解。事实上，这一句就是 max=a>b?(a>c?a:c):(b>c?b:c);即若 a 大于 b，则取 a、c 中的大者，否则取 b、c 中的大者，去掉小括号也无妨，正是体现了条件表达式的"自右向左"的结合性。当然，加上小括号更好理解些。

3.3.4　逗号运算符和逗号表达式

C 语言提供了一种特殊的运算符：逗号","运算符。

1. 逗号表达式

用逗号运算符将两个及两个以上的表达式连接起来组成的式子称为逗号表达式。

逗号表达式的一般形式如下：

表达式 1，表达式 2，……，表达式 n

逗号表达式的计算过程是：先计算表达式 1，再计算表达式 2，依次计算直到表达式 n，表达式 n 的值是整个逗号表达式的值。

例如：

逗号表达式 30+50，16+8 的值为 24。

又如：

逗号表达式 a=10*5，a*40

先计算 a=10*5 得 a=50，再计算 a*40 得 2 000，则逗号表达式的值为 2 000。

2. 使用说明

（1）一个逗号表达式可以与另一个表达式组成一个新的逗号表达式。

例如：

```
(a=30*50，a+10)，a/5
```

先计算 a=1 500，再计算 a+10 得 1 510（注意 a 仍为 1 500），最后计算 a/5 得 300，即整个表达式的值为 300。

（2）逗号运算符在所有运算符中优先级别最低，逗号运算符的结合方向是"从左至右"。

下面两个表达式的运算结果是不同的：

```
b=(a=10，5*a)
b=a=10，5*a
```

第一式是赋值表达式，将一个逗号表达式的值赋给 b，表达式的值为 50，所以 b 的值为 50；第二式是逗号表达式，它包含一个赋值表达式和一个算术表达式，b 的值为 10，而表达式的值为 50。

（3）并不是任何地方出现的逗号都是逗号运算符。例如，在函数的参数中，逗号的作用往往是作为分隔符使用的。

例如：printf("%d,%d,%d\n",a,b,c);

其中的逗号显然不能看作是逗号运算符。

逗号表达式无非是把若干个表达式“串联”起来。在许多情况下，使用逗号表达式的目的只是想分别得到各个表达式的值，而并非一定需要得到或使用整个逗号表达式的值。

3.3.5 sizeof运算符

C 语言提供了整型、实型、字符型 3 种基本类型，而这些基本类型在计算机中的表示则因系统而异。为了使用户能够了解自己所用系统的各类型的长度，C 语言提供了一个单目运算符 sizeof。

1. sizeof的一般调用形式

sizeof(类型名或变量名）

其中，类型名可以是基本类型名，也可以是其他的构造类型名。

2. sizeof的功能

sizeof 运算给出指定类型在内存中所占的字节数。

【例 3.7】测试各类型在内存中所占的字节数。

```
main()
 {char ch;
  int x;
  float y;
  printf(“char=%d\n”,sizeof(char));
  printf(“char(ch)=%d\n”,sizeof(ch));
  printf(“int=%d\n”,sizeof(int));
  printf(“int(x)=%d\n”,sizeof(x));
  printf(“float=%d\n”,sizeof(float));
  printf(“float(y)=%d\n”,sizeof(y));
 }
```

程序的运行结果如下：

```
 char=1
 char(ch)=1
 int=2
 int(x)=2
 float=4
 float(y)=4
```

3.4 运算符的优先级与结合性

C 语言的运算符范围很宽，几乎所有的基本操作都可以作为运算符处理。在求表达式的值时，应按照 C 语言的各运算符的优先级与结合性进行运算。表 3-3 概括了全部运算符的优先级和结合性规则。同一行的运算符具有相同的优先级，下一行中运算符的优先级低于上一行运算

符的优先级。如果同一行的运算符同时出现在一个表达式中，则按结合性规则计算。

表 3-3　运算符的优先级与结合性

优先级	运算符	分类	结合性
1	（）　[]　→　.		从左至右
2	!　~　++　--　-　*　&　sizeof	单目运算符	从右至左
3	*　/　%	双目运算符	从左至右
4	+　-		
5	<<　>>		
6	<　<=　>　>=		
7	==　!=		
8	&		
9	^		
10	\|		
11	&&		
12	\|\|		
13	?:	条件运算符	从右至左
14	=　+=　-=　*=　/=　%= &=　^=　\|=　>>=　<<=	赋值运算符	从右至左
15	,	逗号运算符	从左至右



习题三

1. 填空题

（1）设有 int x=11；则表达式(x++*1/3)的值是_____。

（2）已知数学表达式 $y=x^2-2x+5$，写出对应的 C 语言表达式_____。

（3）已知 a＝10，则表达式 x=(a=a+b,a–b)的值为_____。

2. 选择题

（1）在 C 语言中，下列运算符的操作数必须是 int 类型的运算符是（　　）。

A. %　　B. /　　C. *　　D. ++

（2）假设所有变量均为整型，则表达式(a=2,b=5,a++,b++,a+b)的值为（　　）。

A. 7　　B. 8　　C. 9　　D. 10

（3）已定义 int m=6,n=5;，则执行 m%=n-1;之后，m 的值是（　　）。

A. 0　　B. 2　　C. 1　　D. 3

（4）已有 int a=1,b=1;则执行 b=(a=2*3,a*5),a+7 之后，a,b 的值是（　　）。

A. 30，37　　B. 6，30　　C. 37，44　　D. 5，8

（5）已有 int x=2,y=1;则执行 x++==y--;结果是（　　）。

A. –1　　B. 1　　C. 2　　D. 0

3. 求下面算术表达式的值

（1）x+a%3*(int)(x+y)%2/4　　设 x=2.5，a=7，y=4.7

（2）(float)(a+b)/2+(int)x%(int)y　　设 a=2，b=3，x=3.5，y=2.5

4. 分析下列程序，写出运行结果

（1）main()

```
{
 int x=1;
 printf("%d   %d   %d\n",++x,x++,x);
 }
```

（2）main()

```
{
 int x=7,y=5,a,b,c;
 a=(--x==y++)?—x:++y;
 b=x++%3;
 c=13%y--;
 printf("a=%d,b=%d,c=%d\n",a,b,c);
 }
```

（3）main()

```
{
 int a,b,x=2,y,z=y=3;
 a=(z>=y>=x)?1:2;
 b=z<=y&&y>=x+1;
 printf("a=%d,b=%d\n",a,b);
 }
```

（4）main()

```
{
 int n=7;
 n+=n=n*=n/3;
 printf("n=%d\n",n);
 }
```

（5）main()

```
{
 int x,y,z;
 x=y=1;z=2;
 y=x++-1;
 z=--y+1;
 printf("x=%d,y=%d,z=%d\n",x,y,z);
 }
```

5. 编程题

（1）定义符号常量 N、M，分别代表 12 和 25，输出 N 和 M 的四则运算值。

（2）已知 i=20、j=20，输出 i++、++j 的值，试比较它们有什么区别。

（3）已知 m=20、n=20，输出 m--、--n 的值，试比较它们有什么区别。



上机实习指导

一、学习目标

本章重点介绍了算术表达式、赋值表达式、逗号表达式、条件表达式、自增/自减运算、求

字节数运算。表达式是编写 C 语言语句的基础，是构造语句不可或缺的基本元素，因此是十分重要的。通过本章的学习，应达到以下要求：

熟练掌握各种运算符的功能、用法、使用规则和结合性；

根据数学表达式及算法描述，熟练地写出 C 语言允许的各种表达式形式；

根据 C 语言表达式，熟练写出表达式的运算顺序；

根据 C 语言表达式，熟练计算出表达式的值。

二、应注意的问题

1. 关于运算符的结合性

C 语言中提供了一些“从右至左”结合的“右结合”运算符，如自增/自减运算、求字节数运算、条件运算和赋值运算。其运算顺序是从右至左进行的。例如，a=b=c=1 000，运算顺序是：将 1 000 赋给变量 a，a 再赋给 b，b 再赋给 c，a、b、c 的值均为 1 000。

2. 关于运算顺序与括号的使用

C 语言运算符均规定了优先级，如括号的优先级最高，而逗号运算符的优先级最低。对于 C 语言的初学者，适当使用括号可以改变运算符的优先级；适当使用括号可以使运算符的运算顺序更清楚。例如，a+b/a–b，的运算顺序是先计算 b/a，然后再加 a，最后减 b；而(a+b)/(a–b)，则先计算 a+b，再计算 a–b，最后完成除法运算。此外，当出现诸如 x+++y 这种运算时，也可以适当使用括号加以限制，以避免人为的“二义性”，例如，（x++）+y 与 x+（++y）是两种不同的运算。



上机实习　运算符及表达式应用

一、目的要求

1. 掌握各种运算符及表达式的应用；
2. 能够读懂和编写简单的计算程序。

二、上机内容

1. 指出下面程序中的错误，并改正。

```
main( )
{
 int a=2;b=3;
 scanf(“%d,%d,%d”,a,b,c);
 c+=a+b;
 printf(“a=%d,b=%d,c=%d”,a,b,c);
```

2. 运行下列程序，观察并分析运行结果。

（1）main()

```
{
  int   a,b;
  a=20;b=30;
  printf(“(a+b)/(a-b)=%d\n”, (a+b)/(a-b));
  printf(“a+b/a-b=%d\n”, a+b/a-b);
}
```

（2）main()

```
{
```

```
    printf("100/200=%d\n", 100/200);
    printf("100/200.0=%f\n", 100/200.0);
}
```

（3）main()

```
{
  int  i,j;
  i=100;j=200;
  printf("(i++)+j=%d\n", (i++)+j);
  i=100;j=200;
  printf("i+(++j)=%d\n", i+(++j));
  i=100;j=200;
  printf("i+++j)=%d\n", i+++j);
 }
```

（4）main()

```
 {
   int  i,j,k,m;
   i=j=k=100;
   i=i+1;
   ++j;
   k+=1;
   printf("i=%d\n",i);
   printf("j=%d\n",j);
   printf("k=%d\n",k);
 }
```

（5）调试并运行书中【例 3.7】的程序。

3. 完善程序。

根据题目要求，在横线处填写合适的内容，使程序完整并能正确地上机运行。

（1）计算 x^5+10^5

```
#include  _____
main()
{
long  y,z;
int  x;
x=8;
y=pow(x,5)+_____;
printf("x^5+10^5=%ld\n",y);
}
```

（2）利用条件表达式计算：当 x> y 时 a=10*x，否则，a=10*y

```
main()
{
  int  x,y,a,b;
  x=20; y=10;
  b=_____ ;
  a=10*b;
  printf("a=%d\n",a);
}
```

第 4 章　数据的输入与输出



C 语言本身不提供输入/输出语句，只提供输入/输出函数，如：字符输入/输出函数 getchar()与 putchar()、按格式输入/输出函数 scanf()与 printf()等，这一特点正好符合结构化程序设计的需要，调用这些函数就可实现数据的输入/输出操作，既规范又方便。这一章所涉及到的程序举例，都属于顺序结构程序设计。

【本章要点】

1. 字符输入/输出函数的使用；
2. 按格式输入/输出函数及格式字符的用法。

【学习目标】

1. 要掌握对单个字符进行输入/输出的程序设计；
2. 了解怎样才能做到字符的连续输入/输出；
3. 要熟练掌握按格式输入/输出函数中格式字符的用法，特别是输出格式的控制；
4. 进一步熟悉简单的程序设计——顺序结构的程序设计方法。

【课时建议】

讲授 3 课时，上机 2 课时。

4.1　字符输入/输出函数

4.1.1　字符输入函数getchar()

此函数的作用是从终端（一般默认为键盘，或为系统隐含指定的其他输入设备）上接收一个字符的输入。它的返回值为一整型数，即被输入字符的 ASCII 码值。函数 getchar()没有参数，其一般形式为：

```
getchar()
```

说明：

（1）在用使 C 语言标准 I/O 库中的函数时，应在程序前加上预编译命令#include <stdio.h>加以说明（有关预编译命令的使用，详见第 9 章）。

（2）getchar()只能接收一个字符，而且必须敲回车键后，函数才能接收。

（3）该函数得到的字符可以赋给一个字符变量或整型变量，也可以作为表达式的一部分不赋给任何变量直接参加运算；

（4）它不能单独作为一个语句，一般情况下，要先定义一个字符类型的变量，然后再引用 getchar()函数，并将函数值赋给这个字符型变量。

请看程序举例：

【例 4.1】从键盘接收一个字符并输出该字符和其 ASCII 码值的程序。

```
# include   <stdio.h>
main()
{
  char c;
  c=getchar();
  printf("%c %d",c,c);
}
```

若输入：a↙

打印出：a 97

当程序运行遇到 getchar()函数时，等待接收字符，且输入字符（在屏幕上显示）后，按了回车键程序才能继续运行。Turbo C 还提供了一个与此功能相同的函数 getch()。它与 getchar()不同的是：接收字符时不等待按回车键，且屏幕上不显示输入字符。

【例 4.2】从终端打入两个字符给变量 c1 和 c2，将较大者放到 c，并输出较大者。

编写这样的程序，很容易让我们想到【例 3.6】。该程序是通过 scanf()函数从键盘接收 3 个实型数求最大值输出，现换成从键盘接收两个字符求最大值输出，程序如下：

```
#include 〈stdio.h〉
main()
{
char c,c1,c2;
(c1=getch())>(c2=getch())?(c=c1):(c=c2);
printf("%c",c);
}
```

若输入：fm

则输出：m

请读者思考为什么输出 m，并且考虑将 getch()函数换成 getchar()函数，执行时会有什么不同。

4.1.2 字符输出函数putchar()

putchar 函数的作用是向终端（默认为显示器）输出一个字符，其一般形式为:

```
putchar(ch)
```

该函数是一个有参函数，参数 ch 通常为字符型变量、字符码（0~255 之间的整型变量）或字符本身（字符本身要用单引号括起来），其函数类型是整型。

【例 4.3】从键盘接收一个字符并输出该字符的程序。

```
# include <stdio.h>
main()
{
  char c;
  c=getchar();
  putchar(c);
}
```

输入：a ↙

输出：a

说明：

（1）在使用 putchar 时，也应在程序前使用预编译命令:#include〈stdio.h〉。

（2）在例中，getchar()和 putchar()组合起来使用，输入一个字符赋值给变量 c，然后将该字

符输出到屏幕终端上。

（3）putchar 函数也可以输出控制字符和其他转义字符，如：putchar('\n')输出一个换行符，putchar('\')输出一个单引号字符。

【例 4.4】输出字符和控制字符的程序。

```
#include 〈stdio.h〉
main()
{
 int  b;
 char c;
 b=108; c='m';
 putchar(b); putchar('\n');
 putchar(c); putchar('\n');
}
```

输出为：l
m

【例 4.5】输出字符、控制字符和转义字符的程序。

```
#include 〈stdio.h〉
main()
{
 char a,b;
 a='\'; b='m';
 putchar(a); putchar(b); putchar(a); putchar('\n');
 putchar(b); putchar('\n');
}
```

输出为：'m'
m



注意

通过【例 4.1】与【例 4.4】的对比，请分析用 putchar 函数在输出字符时，与用 printf 函数输出字符有什么不同。

4.1.3　字符的连续输入/输出

在进行输入/输出中，有时要进行字符的连续输入/输出，这里给出利用循环语句 for 和 while 来实现的一种方法，程序的详细分析可在讲了第 5 章以后再进行。

首先，使用 for 语句的无限循环 for(;;)可以达到目的。

【例 4.6】程序 1。

```
# include 〈stdio.h〉
main()
{
 for(;;)
 putchar(getchar());
}
```

输入:abcd↙
输出:abcd
输入:1234↙

```
输出:1234
```

在这个程序里，可以连续输入任何字符，而且这些字符都能输出。这些字符要以行为单位进行处理，输出也是在行终止符（回车）后进行的。这个程序一直处于执行状态，如何使该程序退出执行状态呢？当然可以强制终止程序的执行，但不一定很理想。因此，我们可利用 while 语句的条件表达式设置终止 getchar()函数输入的终止符。

【例 4.7】程序 2。

```
  # include <stdio.h>
  main()
  {
    int c;
    while((c=getchar())!='x')
    putchar(c);
  }
输入:ab321x
输出:ab321
```

该程序一旦发现输入终止符 x 便结束运行，当然还有其他方法终止程序执行，在此不做过多的阐述。

4.2 按格式输入/输出函数

4.2.1 按格式输出函数printf()

printf()函数，在前面各章节中已用到，它的作用是向终端按格式要求输出若干任意类型的数据。

1. 一般形式

```
printf( 控制参数,输出参数 );
```

这是一个带参数的函数，其中的“控制参数”是用双引号括起来的字符串，也称“转换控制说明”，它规定了输出参数各项的输出形式，它包括 3 种信息：一是提示字符串，可原样输出；二是格式转换控制符，由“%”和格式字符组成；三是转义字符，输出一些操作行为。输出参数是需要输出的数据，可以是变量或表达式表列，其项数必须与控制参数中的格式转换控制符个数相同，如图 4-1 所示。

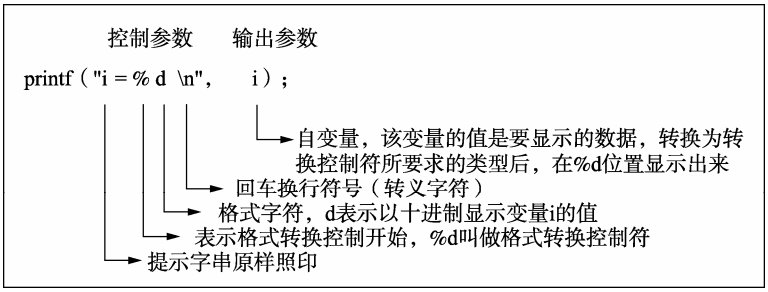


图 4-1 printf()函数的参数说明

如果图 4-1 中 i 的值为 321，则上述语句的输出结果为：

```
i=321
```



说明：

（1）用双引号括起来的是控制参数，i 是输出参数，两者之间必须用逗号隔开；

（2）控制参数中提示字符串如“i=”原样输出；

（3）格式符 d 与%之间不能留有空格；

（4）格式符不能用大写字母；

（5）在控制参数中可以包含“转义字符”，如：“\n”、“\x07”等。其意义在第 2 章已做过介绍。

2. printf()中格式字符的用法

在 printf 函数中，数据的输出形式由格式字符决定。C 语言提供了 9 种格式字符，代表着不同输出的要求（请见表 4-1），如果格式字符使用不当，就会得不到预期的结果。

表 4-1　printf 函数的输出格式符

格式字符	输出形式	示　例	输出结果
d	十进制整数	int a=123;printf("%d",a);	123
o	八进制整数	int a=123; printf("%o",a);	173
x	十六进制整数	int a=123; printf("%x",a);	7b
u	无符号十进制整数	int a=123; printf("%u",a);	123
c	字符	char a='A';printf("%c",a);	A
s	字符串	char *a="OK! ";printf(%s",a);	OK!
f	小数形式实数	float a=123.45; printf(%f",a);	123.450000
e	指数形式实数	float a=123.45; printf(%e",a);	1.234500e+002
g	比 e 和 f 输出宽度较短	float a=123.45; printf(%g",a);	123. 45

各种格式符的具体使用方法分别介绍如下：

（1）d 格式符，以十进制形式输出整数。

① %d，按整型数据的实际长度输出。

② %md，m 代表一个正整数，按 m 指定的宽度输出。若实际数据的位数小于 m，则左端补空格，若大于 m，则按实际位数输出。当 m 前有“-”号时,表示按 m 指定宽度左对齐，右边补空格。

③ %ld，输出长整型数据，且长整型数据必须用该转换控制形式输出。%mld 输出指定宽度的长整型数据。但 int 型数据可用%d 或%ld 格式输出。

【例 4.8】程序 3。

```
main()
{
 int a,b;
 long c,d;
 a=32767; b=1;
 c=2147483647;d=1;
 printf("%4d,%4d\n",a,b);
 printf("%d,%d\n",a,b);
 printf("%ld,%ld\n",c,d);
 printf("%10ld,%10ld\n",c,d);
 }
```

打印出：

```
32767，□□□1（其中□代表一个空格位）
32767,1
2147483647,1
2147483647, □□□□□□□□□1
```

注意

此程序使用的 printf 函数和下面要介绍的 scanf 函数是格式输出/输入函数，与字符输出/输入函数应用一样，应在程序前加上预编译命令#include <stdio.h>加以说明，但因 printf 和 scanf 函数使用频繁，系统允许不需加上此预编译命令。

（2）o 格式符，以无符号八进制形式输出整数。

该格式也可以用%lo 输出长整型，用%mo 输出指定宽度的八进制整数。

【例 4.9】程序 4。

```
main()
{
 int a=-1;
 long b=2;
 printf("%d,%o",a,a);
 printf("%9o,%lo",a,b);
}
```

打印出：

```
-1,177777□□□177777,2
```

（−1 在内存单元中以补码的形式存放，−1 的补码是 1111111111111111，转换为八进制数则为 177777，不会输出带负号的八进制整数。）

（3）x 格式符，以无符号十六进制形式输出整数。

该格式也可以用%lx 输出长整型，用%mx 输出指定宽度的十六进制整数。

【例 4.10】程序 5。

```
main()
{
 int a=-1;
 long b=-2;
 printf("%x,%o,%d\n",a,a,a);
 printf("%lx,%lx\n",a,b);
 printf("%8x",a);
 }
```

打印出：

```
ffff,177777,-1
fffeffff,100cffff
□□□□ffff
```

（4）u 格式符，以无符号十进制形式输出。

无符号型数据也可用%d、%o 或%x 格式输出，有符号 int 型数据也可用%u 格式输出。

【例 4.11】程序 6。

```
main()
{
  unsigned int a=65535;
```

```
    int b=-2;
    printf("%d,%o,%x,%u\n",a,a,a,a);
    printf("%d,%o,%x,%u\n",b,b,b,b);
  }
```

打印出：

```
-1,177777,ffff,65535
-2,177776,fffe,65534
```

（−1 在内存单元中以补码的形式存放，−1 的补码是 1111111111111111，转换为无符号十进制数则为 65535，−2 的补码是 1111111111111110 转换为无符号十进制数则为 65534。）

（5）c 格式符，用来输出一个字符。

对于整数，只要它的值在 0～255 范围内，也可以用字符形式输出，当然一个字符数据也可以转成相应的整型数据（ASCII 码值）输出。%mc 输出指定宽度的字符型数据（占不满时，左补空格）。

【例 4.12】程序 7。

```
  main()
  { char c='b';
    int  i=98;
    printf("%c,%3c,%d\n",c,c,c);
    printf("%c,%3c,%d\n",i,i,i);
  }
```

打印出：

```
b,□□b,98
b,□□b,98
```

（6）s 格式符，用来输出一个字符串。

① %s，输出的字符串原样照印。

② %ms 输出指定宽度的字符串，若实际串长小于 m，则左补足空格，若实际串长大于 m，则按实际串长输出。

③ %−ms 同%ms 一样，不同的是当实际串长小于 m 时，字符串向左边靠，右补空格。

④ %m.ns 输出指定宽度为 m 的、从左端取出的 n 个字符(n 代表一个正整数)。若 n<m 则左补足空格，若 n>m 按 n 个字符输出。

⑤ %−m.ns 同%m.ns 一样，不同的是，当 n<m 时右补足空格。

【例 4.13】程序 8。

```
  main()
  {
    char *a="hello,world";
    printf(":%10s:\n",a);
    printf(":%-10s:\n",a);
    printf(":%20s:\n",a);
    printf(":%-20s:\n",a);
    printf(":%20.10s:\n",a);
    printf(":%-20.10s:\n",a);
    printf(":%.10s:\n",a);
    printf("%s\n","very good");
  }
```

打印出：

```
:hello,world:
:hello,world:
:□□□□□□□□□hello,world:
:hello,world□□□□□□□□□:
:□□□□□□□□□□hello,worl:
:hello,worl□□□□□□□□□□:
:hello,worl:
very good
```

%.10s 指定了 n，未指定 m，则此时 n=m=10；输出字符串"very good"不包括引号。

（7）f 格式符，用来输出实数（单、双精度），以小数形式输出。

① %f 按系统规定的格式输出，即整数部分全部输出，小数部分取 6 位。在一般系统下，单精度实数的有效位数为 7 位（即不包括小数点在内的前 7 位准确无误，超过部分，虽也打印出来，但无意义）,双精度实数的有效位数为 16 位。

② %m.nf（或%−m.nf）是输出指定宽度为 m（小数点也占一位宽度），保留 n 位小数的实数。若输出数实际长度 l<m，则左补空格数字向右边靠（或右补空格，数字向左边靠。当格式符为%0m.n 时，以 0 填充），若 l>m 则按实际长度输出，并保留 n 位小数。

③ %.nf 也是按实际长度输出，保留 n 位小数。

【例 4.14】程序 9。

```
main()
{
  float  f=1111.11111;
  double d=22222.2222222222222;
  printf("%f,%f,%10.2f\n",f,d,f);
  printf("%-10.2f,%010.2f,%.2f\n",f,d,d);
}
```

打印出：

```
1111.111084,22222.222222,□□□1111.11
1111.11□□□,0022222.22,22222.22
```

（8）e 格式符，以指数形式输出实数。

① %e 按系统规定输出指数形式的实数，系统规定：指数部分占 5 位（如 e+003 或 e−003）,小数点占一位，小数点前只有一个非零数字，小数点后占六位，共计占宽度 13 位（如 1.234567e+003）。

② %m.ne 输出指定宽度为 m，保留 n 位小数的实数。若实际宽度 l<m，则左端补空格数字向右边靠（若为%−m.ne 则右端补空格，数字向左边靠），若实际长度 l>m，则按实际长度输出且保留 n 位小数。

③ %.ne 是按实际长度输出，保留 n 位小数。

④ %me 是输出指定宽度为 m 的实数，保留 6 位小数。若实际长度 l>m，则按实际长度输出。

【例 4.15】程序 10。

```
main()
{
  folat  x＝654.321；
  printf("%e,%10e,%10.2e,%.2e,%-10.2e",x,x,x,x,x);
}
```

打印出：

```
6.543210e+002,6.543210e+002,□6.54e+002,6.54e+002,6.54e+002□
```

（9）g 格式符，用来输出实数，输出格式为 f 格式或 e 格式，系统根据数据占宽度 m 大小，自动选择占宽度较小的某种格式输出，g 格式符不输出小数点后无意义的零。

【例 4.16】程序 11。

```
main()
{
  float   x=654.321;
  printf("%f,%e,%g",x,x,x);
}
```

打印出：

```
654.320984,6.543210e+002,654.321（其中输出的 654.320984 是因为在内存中的存储误差引起的）
```

4.2.2　按格式输入函数scanf()

1. scanf()的一般形式

scanf 函数可以用来按格式要求输入任何类型的多个数据。其引用的一般形式为：

```
scanf(控制参数,地址列表)
```

“控制参数”的含义同 printf 函数，地址表列是由若干个地址组成的表列，可以是变量的地址或字符串的首地址。

先看一个使用 scanf 函数的程序例题及说明。

【例 4.17】程序 12。

```
#include <stdio.h>
main()
    {
     int a,b,c;
     char s[4];
     scanf("a=%d b=%d %s",&a,&b,s);
     c=a+b;
     printf("%s=%d+%d=%d\n",s,a,b,c);
    }
```

运行结果:

```
a=2 b=3 a+b↙   /*这行数据都是运行时由键盘输入的*/
a+b=2+3=5      /*这是输出的计算结果*/
```

该例是将整数 2、3 分别输入给变量 a、b，将字符串 a+b 输入给字符串变量 s，然后输出计算 a+b 的结果。

说明：

（1）在 scanf()函数中，由双引号括起来的是控制参数，其中的%d，%s 是格式转换控制符，用来说明键盘输入的数据是整型和字符串型；a=、b=是提示串，为用户输入数据增加必要的提示信息，从而减少输入的错误。但与其他语言不同，程序运行时它并不显示在屏幕上提示你输入相应数据，它也是输入的一部分，必须在与它相对应的位置从键盘敲入。

（2）&a，&b，s 是 scanf()函数的地址表列，"&"是地址运算符，&a 指向 a 在内存中的地址，字符串变量 s 前不许加&，s 指出了字符串变量 s 的首地址。

（3）在给多个输入项输入数据时，键盘输入的各项信息之间可以用空格，TAB 或回车键作

为分隔符，例中输入使用了空格和回车键。

（4）C 语言的初学者，在使用该语句时，非常容易出现丢掉地址运算符"&"的错误。例如：

```
scanf("%d %d ",a,b,);
```

这样使用是不对的，对此读者一定要注意。

2. scanf()中格式字符的用法

scanf 函数是格式输入函数，其输入数据的类型要受格式字符的控制。输入格式转换控制符有八种，如表 4-2 所示。

表 4-2　scanf 函数的输入格式控制符

格式字符	输入要求	示　例	输入数据
d	十进制整数	Scanf("%d",a);	123
o	八进制整数	Scanf("%o",a);	173
x	十六进制整数	Scanf("%x",a);	7b
u	无符号十进制整数	Scanf("%u",a);	123
c	字符	Scanf("%c",a);	A
s	字符串	Scanf(%s",a);	OK!
f	小数形式实数	Scanf(%f",a);	123.450000
e	指数形式实数	Scanf(%e",a);	1.234500e+002

格式符的具体使用方法和应注意的事项，具体举例介绍如下。

（1）d 格式符，用来输入十进制整数。输入时有以下两种情形：

① 格式转换控制符之间有非格式字符。

例如：

```
scanf("i=%d",&i);        若输入 i=123                合法。
scanf("%d,%d",&i,&j);    若输入 123,456              合法。
scanf("%d:%d",&i,&j);    若输入 123:456              合法。
scanf("%d□%d",&i,&j);若输入 123□456 或 123□□□456 合法。
```

该例子说明，如果转换控制说明中有格式转换控制符之外的字符，则输入时，要在与此相对应的部分输入与此相同的字符(当该字符是一个空格时，输入一个或多个空格均合法)。

② 格式转换控制符之间没有非格式字符。

例如：

```
scanf("%d%d",&x,&y);   若输入：1□□2↙              合法。
                       若输入：1↙
                               2↙                  合法。
                       若输入：1（按 Tab 键）2↙    合法。
```

该例子说明，如果转换控制说明中格式转换控制符之间没有非格式字符，则输入时，要在两个数据间打一个或一个以上的空格，或打回车键，或按 Tab 键，都是合法的。

以上说明其他格式符也有类似的情况。

（2）o 格式符，用来输入八进制整数。

【例 4.18】输入两个八进制数，求其积并用八进制数显示的程序。

```
main()
  { int a,b;
    scanf("%o□%o",&a,&b);
```

```
        printf("%o*%o=%o\n",a,b,a*b);
      }
若输入：12□23↙
则输出：12*23=276
```

该例中输入数据与输出结果都是八进制数，看上去与十进制数没有区别，请读者分析为什么会这样，如果输入 12 和 24 结果会如何？

（3）x 格式符，用来输入十六进制整数。

【例 4.19】输入一个十六进制数，并分别用十六进制数和十进制数显示的程序。

```
    main()
    { int a;
      scanf("%x",&a);
      printf("%x\n",a);
      printf("%d\n",a);
     }
若输入：abc↙
则输出：abc
        2748
```

（4）c 格式符，用来输入单个字符。在输入时，空格和转义字符都会按有效字符接收；当格式控制符之间有空格时，输入的空格将被忽视。请见【例 4.20】程序 13～【例 4.23】程序 16。

【例 4.20】程序 13。

```
    main()
    { char a;
      scanf("%c",&a);
      printf("%c\n",a);
    }
输入：e↙
输出：e
输入：good↙
输出：g      （仅第一个字符 g 赋给变量 a）
```

【例 4.21】程序 14。

```
    main()
    {
      char a,b,c;
      scanf("%c%c%c",&a,&b,&c);
      printf("%c%c%c\n",a,b,c);
    }
输入：top↙
输出：top
输入：top↙
输出：to    （空格赋给了变量 b）
```

【例 4.22】程序 15。

```
    main()
    { char a,b;
      scanf("%c   %c",&a,&b);
      printf("%c%c\n",a,b);
    }
```

```
输入：we↙
输出：we
输入：w      e↙
输出：we
输入：w↙
      e↙
输出：we
```
（在这种情况下，忽视了空格）

【例 4.23】程序 16。

```
main()
{ char a,b;
  scanf("%c",&a);
  scanf("%c",&b);
  printf("%c%c\n",a,b);
}
输入：ask↙  （字符 a 赋给了变量 a，字符 s 赋给了变量 b）
输出：as
输入：a↙    （字符 a 赋给了变量 a，字符↙赋给了变量 b）
输出：a
输入：a b↙  （字符 a 赋给了变量 a，字符空格赋给了变量 b）
输出：a
```

其他形式，请读者上机自已试验。

（5）s 格式符，用来输入字符串，将字符串送到一个字符数组中，在输入时以非空白字符开始，以第一个空白字符结束。字符串以串结束标志'\0'作为其最后一个字符，系统对字符串常量自动加一个‘\0’作为结束标志，若“abcdefg”共有 7 个字符，但在内存中占八个字节，最后一个字节放‘\0’。

（6）f 格式符，用来输入实数，可以用小数形式或指数形式输入。

（7）e 格式符与 f 作用相同，e 与 f 可以互相替换。

（8）其他相关说明：

① 格式字符前面的 l（有 5 种情况,如：%ld,%lo,%lx,%lf,%le），是表示读长整型数据或 double 型数据。例如：

```
long i;
double j;
scanf("%ld,%lf",&i,&j);
```

② 格式字符前面的数字，指定输入数据所占宽度（但不能像 printf()函数那样指定小数位数），当输入满足格式要求宽度，或遇到空白字符、非法字符时，将结束数据输入。

例如：

```
scanf("%2d%d",&i,&j);
```

若输入 123　此时变量 i 存入 12,3 存入 j 中

再如：scanf("%2d%4d",&i,&j);

若输入 123　456　此时变量 i 中存入 12，3 存入 j 中，456 不能读入内存变量之中。

若输入 123456，此时变量 i 中存入 12，3456 存入变量 j 中。

③ 格式字符前面的 h（有 3 种情况,如：%hd,%ho,%hx），用于输入短整型数据。

④ %后面的*，表示本输入项在读入后不赋给任何相应的变量。例如：

```
scanf("%2d%*3d%3d",&i,&j);
```



若输入 123 456，此时变量 i 中存入 12，3 被读入，但不存入 j 中，而 456 存入了变量 j 中。

⑤ 为输入方便需要屏幕显示提示输入信息，可在输入语句之前加上一个输出“提示信息”的语句。

3. scanf()的应用综合举例

【例 4.24】程序 17。

```
main()
{
 char     c;
 int      year,month,day;
 float    f;
 double d;
 char     s[10];
 printf("\nInput a char & 3 Integer(Decimal,Octal,Hexdecimal):\t");
 scanf("%c %d %o %x",&c,&year,&month,&day);
 printf("\nD=%d,O=%o,H=%x,C=%d\n",year,month,day,c);
 printf("\nInput a Float & a Double:\t");
 scanf("%f %lf",&f,&d);
 printf("\nFloat=%e,Double=%lf\n",f,d);
 printf("\nInput a String of chars:\t");
 scanf("%10s",s);
 printf("\nString=%s\n",s);
 printf("\nInput present date & week (year,month,week,day):\t") ;
 scanf("%4d %2d %*s %2d",&year,&month,&day);
 printf("\nPresent date:%d%d%d\n",year,month,day);
}
```

输出结果与输入数据：

```
Input a char & 3 Integer(Decimal,Octal,Hexdecimal): c 1993 7 19↙
D=1993,O=7,H=19,C=C
Input a Float & a Double:        123.456    1.23456789↙
Float=1.234560e+002,Double=1.234568
Input a String of chars:         abcdefg↙
strint=abcdefg
Input present date & week (year,month,week,day):    1993 7 abc 19↙
Present date:1993719
```

【例 4.25】程序 18。

```
main()
   {
      static char s[]="ABCDEFGHIJKLMN";

           double x=1.2345678;

      printf("r1:%20.10s\n",s);
      printf("r2:%-20.10s\n",s);
      printf("r3:%f\n",x);
      printf("r4:%10.2f\n",x);
      printf("r5:%-10.2f\n",x);
```

```
    printf("r6:%010.2f\n",x);
    printf("r7:%e\n",x);
    printf("r8:%g\n",x);
  }
```

打印出：

```
  r1:□□□□□□□□□□ABCDEFGHIJ
  r2:ABCDEFGHIJ□□□□□□□□□□
  r3:1.234568
  r4:□□□□□□1.23
  r5:1.23□□□□□□
  r6:0000001.23  （注：当格式符为%0m.n 时，输出数实际长度小于 m 以 0 填充）
  r7:1.234568E+000
  r8:1.234568
```



习题四

1. 单项选择题

（1）下列叙述不正确的是_____。

A. getch ()用于从键盘上键入一个字符，不必按回车键

B. 多个输入数据项的分隔字符可用空格、TAB、换行、数字 0 等

C. 结束数据输入方法有：遇到空白字符、非法字符或输入满足了格式要求长度

D. Scanf 函数的地址表列中，除 s 外其他变量前必须加地址运算符“&”

（2）当输入数据为 12345678 时，下面程序运行的结果是_____。

```
main()
{ inta,b;
  Scanf("%2d%*2s%2d",&a,&b);
  Printf("%d\n",a+b);
 }
```

A. 46　　B. 57　　C. 68　　D. 出错

（3）putchar 函数可以向终端输出一个_____。

A. 字符串　　B. 整型变量的值　　C. 实型变量的值　　D. 字符或字符型变量的值

（4）按格式输出的格式符_____。

A. 可以与%之间有空格　　B. 可以用大小写字母

C. 可以控制输出任意类型的数据　　D. 可以直接控制输出小数的位数

2. 回答问题

（1）已知变量 a=12,b=24,c='A',d='a',e=2.58,f=–2.58,g=68.5 和语句

```
scanf("%6d,%6d,%c%c%f%f%*f%f",&a,&b,&c,&d,&e,&f,&g);
```

如何在键盘上输入数据？

（2）给出如下程序，写出该程序的意义，若读入 a=b=5，运行结果如何？

```
main()
{
 int a,b,sum;
```

```
    printf("type a number please! a=?\n");
    scanf("%d",&a);
    printf("type another please! b=?\n");
    scanf("%d",&b);
    sum=a+b;
    printf("a sum b is %d\n",sum);
  }。
```

3. 写下列程序的执行结果。

（1）

```
main()
{
 int x,y,z;
 long m,n,o,;
 unsigned p,q,r;
 x=32766; y=1; z=2;
 m=2147483646; n=1; o=2;
 p=65534; q=1; r=2;
 printf("%d,%d\n",x+y,x+z);
 printf("%ld,%ld\n",m+n,m+o);
 printf("%u,%u",p+q,p+r);
}
```

（2）

```
main()
{
 char c1,c2,c3,c4;
 c1='y';c2='e';c3='s';c4=', ';
 a="I am computer.";
 printf("%c%c%c%c",c1,c2,c3,c4);
}
```

（3）

```
main()
{
char x,y;
x='a'; y='b';
printf("pq\brs\ttw\r");
printf("%c\\%c'\n",x,y);
printf("%o\n","\123');
}
```

（4）

```
main()
{
 float x=58.8873, y=-555.678;
 char c='B';
 long n=7567890;
 unsingned u=76768;
 printf("%f,%f\n",x,y);
 printf("%-12f,%-12f\n,"x,y);
```

```
    printf("%8.3f,%8.3f,%.3f,%.3f,%5f,%3f\n",x,y,x,y,x,y);
    printf("%e,%10.2e\n",x,y);
    printf("%c,%d,%o,%x\n",c,c,c,c);
    printf("%ld,%lo,%x\n",n,n,n);
    printf("%u,%o,%x,%d\n",u,u,u,u);
    printf("%s,%5,3s\n","COMPUTER","ABCDEFGHI");
  }
```

5. 编写程序

（1）以键盘输入一个大写字母，要求改用小写字母输出（提示：大写字母对应的 ASCII 码序号比相应的小写字母的 ASCII 码序号小 32）。

（2）从键盘上输入学生的 3 门课程成绩，求其总成绩、平均成绩和总成绩除 3 的余值。

（3）若 a=1,b=2,c=3,d=7.2,e=−5.5,f=1.56,g=12345,h=123456,i='0',j='p'设计一个程序，得到下列输出结果。

a=1　b=2　c=3

d=7.200000,e=−5.500000,f=1.560000

d+e=−1.70　　e+f=−3.94　d+f=8.760

g=12345　　h=123456

i='0' or 11

j='p' or 112



上机实习指导

一、学习目标

数据的输入、处理、输出是任何程序都不可缺少的基本功能。本章主要介绍了 4 个输入/输出函数，C 语言通过调用这些函数而实现数据的输入/输出操作。因此，学好本章内容是学好 C 语言程序设计的基础。通过本章学习，应达到如下要求：

1. 掌握各个输入/输出函数的使用方法；

2. 掌握程序输出格式的控制。

二、应注意的问题

1. 注意使用预编译命令#include

当使用函数 getchar 和 putchar 时，应在程序前使用预编译命令：

```
#include <stdio.h>
```

该命令的作用是把头文件 stdio.h 包含到该程序中。这是因为 getchar 和 putchar 都是库函数，它们的原型说明（如函数的类型及其参数的类型、函数中用到的符号常量等等）都放在头文件中，如果不将其包含进去，程序将不能顺利通过编译。

2. 使用 scanf()函数时容易出现的错误

（1）变量名前不要忘记写取地址运算符&

当在程序中使用 scanf()函数时，如果在变量名前忘记了写取地址运算符&（如：scanf("%d",a);），程序编译和连接都能顺利通过，但运行结果一定不对。这种错误很难发现，所以要特别注意。

（2）输入实型数据时不要指定小数位数

使用 printf()函数输出实型数据时，可以指定数据的宽度和小数位数；但使用 scanf()函数输

入数据时，允许指定数据的宽度，却不允许指定小数位数。

看下面的程序：

```
main()
{
   float a;
   scanf("%.2f",&a);
   printf("a=%f",a);
}
```

该程序同样能顺利通过编译和连接，但运行结果却是错误的。运行时，不等待输入变量 a 的值，就直接显示 a=12…的一串乱数。

（3）不能用 scanf()函数的参数来显示输入数据的提示信息

例如：

```
scanf("a=%d,b=%d",&a,&b);
```

该语句的设计者是想在程序运行时，先显示“a=”和“b=”的提示信息，然后再输入变量 a 和 b 的值。这种想法很好但做法是错误的，达不到该目的。因为在 scanf()函数的控制参数中，格式转换控制符以外的所有字符，运行时要求原样输入而不是原样输出。

要想达到上述目的，应该用输出函数帮助实现，如采用语句：

```
printf("a=");      （或用 printf("输入第一个整数");）
scanf("%d",&a);
printf("b=");      （或用 printf("输入第二个整数");）
scanf("%d",&b);
```



上机实习一　字符的输入/输出

一、目的要求

1. 掌握 getchar()和 putchar()函数的使用方法；
2. 了解 getch()函数与 getchar()函数使用时的不同点。

二、上机内容

1.运行下列程序，注意观察运行结果

（1）

```
#include <stdio.h>
main()
{
   char c1,c2;
   clrscr();
   c1=getchar();
   c2=getchar();
   printf("c1=\'%c\'\n",c1);   /* "\' "作用是输出单撇号"' " */
}
```

说明：

运行该程序时，分别输入下面 3 组数据，注意观察各自的运行结果，并思考为什么。

第一组数据：

```
a ↙
```

```
第二组数据：
        ab ↙
第三组数据：
        abc ↙
```

（2） #include <stdio.h>

```
main()
{
   char ch,c1,c2;
   clrscr();
   printf("请输入一个字母：");
   ch=getchar();
   c1=ch-1;
   c2=ch+1;
   printf("\n%c 的前一个字母是%c,后一个字母是%c",ch,c1,c2);
}
```

说明：

可将程序中的语句“ch=getchar();”改成“ch=getch();”，看运行结果有什么变化。

2. 完善程序

下面程序的功能是，输入一个小写字母，输出其对应的大写字母；若输入的不是小写字母，则提示输入出错。

请在程序中的横线处填写正确的语句或表达式，使程序完整；并上机调试，使其运行结果与下面给出的结果一致。

```
#include <stdio.h>
main()
{
   char ch1,ch2;
   clrscr();
   printf("请输入一个小写字母：");
   ch1=_____;
   ch2=_____;
   (_____)?putchar(_____):printf("输入出错! ");
}
```

运行结果 1：

```
请输入一个小写字母：
a ↙
A
```

运行结果 2：

```
请输入一个小写字母：
# ↙
输入出错!
```

运行结果 3：

```
请输入一个小写字母：
d ↙
D
```



上机实习二　格式输入/输出函数的使用

一、目的要求

1. 掌握 scanf()和 printf()函数的调用方法及允许使用的格式转换控制符；
2. 熟悉 scanf()函数对各种类型数据的键盘输入格式的要求；
3. 能灵活应用 printf()函数控制屏幕输出格式。

二、上机内容

1.运行下列程序，注意观察运行结果

（1）main()

```
{
 int a,b;
 float c;
 clrscr();
 scanf(”%d%d%f”,&a,&b,&c);
 printf(”a=%d\n”,a);
 printf(”b=%d\n”,b);
 printf(”c=%f\n”,c);
}
```

说明：

运行这个程序时，按下面 4 种格式输入数据，注意观察各运行结果。

第一种输入格式：

```
10 25 4.72 ↙
```

第二种输入格式：

```
10 ↙
25 ↙
4.72 ↙
```

第三种输入格式：

```
10,25,4.72 ↙ （这种输入格式对吗？想一想为什么）
```

（2）main()

```
{
 int a,b;
 char c;
 clrscr();
 scanf(”%d,%d;%c”,&a,&b,&c);
 printf(”a=%d\n”,a);
 printf(”b=%d\n”,b);
 printf(”c=%c\n”,c);
}
```

说明：

运行程序时，如果想使变量 a 的值为 113，b 的值为 3270，c 字符为'x'，那么该如何从键盘输入数据呢？

（3）main()

```
{
 float p=3.14159;
 clrscr();
 printf(”p=%.2f\n”,p);
 printf(”p=%.4f\n”,p);
 printf(”p=%10.2f\n”,p);
 printf(”p=%10.4f\n”,p);
 printf(”p=%-10.2f\n”,p);
 printf(”p=%-10.4f\n”,p);
}
```

说明：

分析该程序的运行结果并运行观察，然后在语句

```
clrscr();
```

的后面加入语句

```
printf(“\n\n\n”);
```

再次运行程序，注意观察结果有什么变化。

2. 完善程序

下面程序的功能是，根据商品的原价和折扣率，计算商品的实际售价。请在程序中的横线处填写正确的语句或表达式，使程序完整。上机调试，使程序的运行结果与给出的结果一致。

```
main()
{
 float cost,percent,c;
 clrscr();
 printf("请输入商品的原价（单位：元）");
 scanf("_____",&cost);
 printf(_____);
 scanf("_____",&percent);
 c=cost*percent;
 printf(_____,c);
}
```

运行结果：

```
请输入商品的原价（单位：元）120 ↙
请输入折扣率：0.85 ↙
实际售价为：102.00 元
```

3. 编写程序

已知：1 公里=2 里=1 000 米

设计一个程序，其功能为：能将输入的公里数换算成里和米。要求输入/输出形式如下：

```
请输入公里数：2.5 ↙          （以数据 2.5 为例）
2.5 公里=5.00 里=25000.00 米
```

该题编写好后，如果上机时间允许，请上机调试通过。

第 5 章　控制结构程序设计



语句是 C 程序的基本成分，控制语句可以构成程序的控制结构，使程序按特定次序执行，是程序设计中算法实现的重要手段。本章主要介绍各种控制语句及相关的表达式以及含有控制结构的程序设计方法。

【本章要点】

1. 关系表达式与逻辑表达式；
2. 条件选择语句 if 的 3 种形式及应用程序举例；
3. 开关语句 switch；
4. 循环语句 for、while 和 do-while 的特点及循环程序设计；
5. 中止语句 break、继续语句 continue 及无条件转移 goto。

【学习目标】

1. 掌握分支语句的使用方法和分支程序设计的方法；
2. 掌握循环语句的使用方法和循环程序设计的方法；
3. 掌握转向语句的使用方法；
4. 理解程序设计中的几种常用算法的基本思想。

【课时建议】

讲授 10 课时，上机 6 课时。

5.1　关系表达式与逻辑表达式

关系表达式与逻辑表达式是构成程序控制结构的关键元素，控制结构包括选择结构和循环结构等，在这些结构中要靠表达式的值来判断程序的走向。因此，在理解和熟悉程序设计的常用算法时，掌握关系表达式与逻辑表达式的用法要比掌握其他表达式更为重要。

5.1.1　关系运算符与关系表达式

1. 关系运算符

c 语言有 6 种关系运算符：<(小于)、<=(小于或等于)、>(大于)、>=(大于或等于)、==(等于)、!=(不等于)。

说明：

（1）<、<=、>、>=的优先级相同，且高于==、!=的优先级；==、!=的优先级相同；

例如：x==y>z 等效于 x==(y>z)；z>x−y 等效于 z>(x−y)；x=y<z 等效于 x=(y<z)。

（2）关系运算符的优先级低于算术运算符，高于赋值运算符，使用时要注意区分==（关系运算符）与=（赋值运算符）的不同。

2. 关系表达式

用关系运算符将两个表达式(可以是算术表达式、赋值表达式、字符表达式、关系表达式、逻辑表达式)连接起来的式子，称关系表达式。

例如：(x>y)>z−5 和'x'>'y'都是关系表达式。

说明：

（1）在关系运算中，表达式只有真或假两个值，输出时用 1 和 0 表示。

（2）若规定的关系成立，则其结果为 1，反之为 0；1 和 0 总是 int 型的，并执行通常的算术转换。

（3）c 语言没有布尔量，这一点与其他语言不同。例如：若 x=3，y=2，z=1 则：x>y 的值为 1，代表“真”；(x<y)==z 的值为 0，代表“假”。

（4）关系表达式的值反映了两个表达式比较和判断的结果：一种是判断条件正确，命名为“真”；另一种是判断条件不正确，命名为“假”。这种表达式常常作为判断条件应用于条件选择语句中。

5.1.2 逻辑运算符与逻辑表达式

1. 逻辑运算符

在 c 语言中，有类似于其他语言中使用的 AND（与）、OR（或）、NOT（非）的三种逻辑运算符：&&(逻辑与)、||(逻辑或)、!(逻辑非)。

说明：

（1）三种运算符的优先级顺序为：!(非)→&&(与)→||(或)，即“!”为最高。

（2）逻辑运算符中的“&&”和“||”低于关系运算符，“!”高于算术运算符。

（3）“&&”和“||”是双目运算符，它要求有两个运算量(操作数)，如：x&&y，(x>y)||(a<b)；在运算符两边的操作数不一定是同类型的，但其中每一个都必须是基本型或指针型，结果为整型。

（4）“!”是单目运算符，只要求有一个运算量。如：!x，!(x<y)。

（5）逻辑运算的规则是：逻辑与运算时，两者都为真结果才为真；逻辑或运算时，只要一个为真结果就为真；逻辑非运算时，非真即为假、非假即为真。表 5-1 为逻辑运算的“真值表”，请看 x 和 y 参与各种逻辑运算所得到的值。

表 5-1 逻辑运算的“真值表”

x	y	!x	!y	x&&y	x\|\|y
真（非 0）	真（非 0）	假（0）	假（0）	真（1）	真（1）
真（非 0）	假（0）	假（0）	真（1）	假（0）	真（1）
假（0）	真(非 0)	真（1）	假（0）	假（0）	真（1）
假（0）	假（0）	真（1）	真（1）	假（0）	假（0）

2. 逻辑表达式

用逻辑运算符将关系表达式或逻辑量连接起来就是逻辑表达式。逻辑表达式的值应该是一个逻辑量“真”或“假”。C 语言编译系统在给出逻辑运算结果时，以数值 1 代表“真”，以 0 代表“假”；在判端一个逻辑量时，以非 0 代表“真”。

例如：

（1）若 x=5，y=6，则：

!x 的值为 0　　（x 不为 0 是真，非真即为假，结果是“假”）
x&&y 的值为 1　　（x 不为 0 并且 y 也不为 0，两者都为真结果为“真”）
x||y 的值为 1　　（x 或者 y 有一个不为 0，则值不为 0 结果为“真”）
!x||y 的值为 1　　（先求出非 x 的值为 0，再与不为 0 的 y 或运算，结果为“真”）
x&&2||!y 的值为 1　（非 y 为 0，x 与 2 不为 0，两者或运算，结果为“真”）

（2）5>2&&2||6<4-!0 其值为 1 代表“真”。

该例的运算中其优先级顺序如下：

“!”运算符高于算术运算符“-”；“-”运算符高于关系运算符“>”或“<”；“>”或“<”高于逻辑运算符“&&”和“||”；而“&&”和“||”同级，从左向右结合。

① !0 运算得 1（真）
② 4–1 运算得 3
③ 5>2 运算得 1(真)
④ 6<3 运算得 0(假)
⑤ 1&&2||0 运算得 1，结果为“真”。

5.2　条件选择语句if

在学习了关系表达式和逻辑表达式以后，就可以进行比顺序结构要复杂一点的程序结构设计，如选择结构和循环结构等。选择结构分为条件选择与多分支选择，本节所涉及的是构成条件选择结构的条件选择语句 if 语句。

5.2.1　if语句的三种形式

1. if语句的第一种形式

if(表达式)　语句

其中“表达式”一般为逻辑表达式或关系表达式，可以是任意的数据类型(整型、实型、字符型、指针型)。例如：

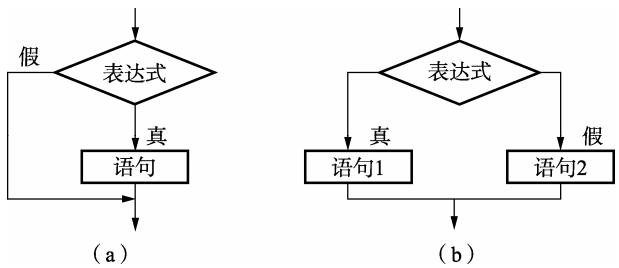
（1）if(a>b && a>c)　printf("%d"，a)；

其中的表达式是逻辑表达式。

（2）if('a')　printf("%d"，'b')；

其中表达式的类型为字符型，最后的执行结果是输出“b”的 ASCII 码值 98。

系统对表达式的值进行判断，若为 0，按“假”处理，若为非 0，按“真”处理，执行指定的语句。流程图见图 5-1(a)。



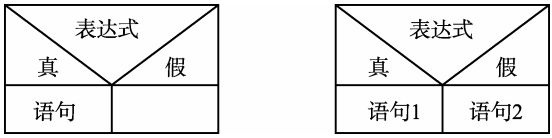


图 5.1 流程图

2. if语句的第二种形式

```
if(表达式 )  语句 1
else  语句 2
```

其中表达式同上，流程图见图 5-1(b)。例如：

```
if(a>b)  printf("%d"，a);
else  printf("%d"，b)；
```

3. if语句的第三种形式

```
if(表达式 1)  语句 1
else  if(表达式 2)  语句 2
else  if(表达式 3)  语句 3
  .                    .
  .                    .
  .                    .
else  if(表达式 n)  语句 n
else                  语句 n+1
```

其中表达式同上，流程图见图 5-2。

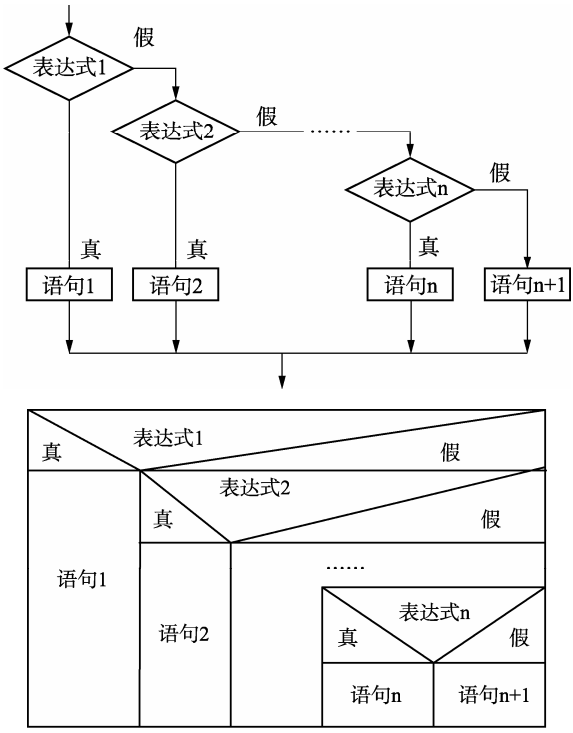


图 5-2 流程图

该语句的程序段写法如下例：

```
if(score>89)        grade='5';
else   if(score>74)grade='4';
else   if(score>59)grade='3';
else                grade='2';
```

说明：

（1）在第二种、第三种形式的 if 语句中，在每个语句后都有一个分号，整个语句结束处有一分号。每个分号都是必不可少的，否则将出现语法错误。但不要误认为上面是用分号隔开的若干语句，它们仍属于同一个 if 语句。

（2）在 if 和 else 后面可以只含一个内嵌的操作语句，也可以有多个操作语句，此时要用花括号“{　}”将几个语句括起来组成一个复合语句。注意在复合语句右括号后面不要再写分号。如：

```
if(a>b)
   {x=a；y=b；}
else
   {x=b；y=a；}
```

5.2.2　条件分支程序设计举例

1. 比较简单的分支程序，if中只用单语句

【例 5.1】用 if 语句编写比较 a、b 两个数的大小，且把大者赋给 L 的程序如下：

```
main()
   {
   int   a，b，L；
   a=3；b=4；
   if(a>b)      L=a；
   else          L=b；
   printf("L=%d\n"，L)；
   }
```

该例题中，因为 a 值为 3，b 值为 4，关系式 a>b 不成立，其值为 0，结果执行 else 后的语句 L=b。执行结果是：

```
L=4
```

2. if语句中带有复合语句的程序

在 C 语言里凡是能用简单语句的地方都可以使用复合语句。因此 if 语句中的语句 1 和语句 2 都可以是复合语句，请看下面程序举例：

【例 5.2】编写比较 a、b 两个数的大小，且把大者赋给 x，小者赋给 y 的程序。

```
main( )
   {
   int   a，b，x，y；
   a=3；   b=4；
   if(a>b)
   {x=a；y=b；}
   else
   {x=b；y=a；}
   printf("x=%d   y=%d"，x，y)；
   }
```

执行结果：

```
x=4  y=3
```

3. 多分支条件语句程序

【例 5.3】编写一程序，根据学生的考分来划分成绩的优、良、及格和不及格。优、良、及格、不及格，我们分别用 5、4、3、2 四个等级来表示，且按如下规定划分:

分数	等级
100～90	5
89～75	4
74～60	3
59～0	2

假定考分变量为 score，学号变量为 snum，则其 C 程序如下：

```
main( )
  {
  int  score，snum；
  char  grade；
  scanf("%d，%d"，&snum，&score)；
  if(score>89)
    grade='5'；
  else  if(score>74)
          grade='4'；
        else  if(score>59)
                grade='3'；
              else
                grade='2'；
  printf("%d：%c\n"，snum，grade)；
  }
```

假定 3 号学生考分为 83 分。待程序执行后，我们从键盘上敲入“3,83↙”，终端屏幕上就会显示出该程序的运行结果：

```
3：4
```

表示 3 号学生成绩为良。

4. 带有嵌套的if语句程序

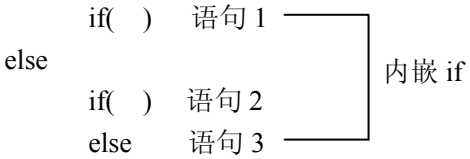
说明：

（1）所谓带有嵌套的 if 语句，是指在 if 语句中又包含一个或多个 if 语句。其一般形式为：

```
if(  )
  if(  )  语句 1 ┐
  else    语句 2 ┘ 内嵌 if
else
  if(  )  语句 3 ┐
  else    语句 4 ┘ 内嵌 if
```

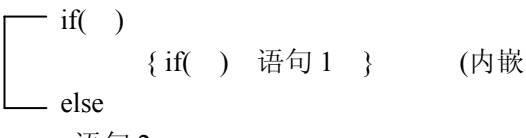
（2）应该注意 if 与 else 的配对关系：else 总是与它上面最近的 if 配对。假如写成：

```
if(  )
```



编程序者把第一个 else 写在与第一个 if(外层 if)同一列上，希望它与第一个 if 匹配， 但实际上是与第二个 if 配对， 因为它们距离最近。因此最好使内嵌 if 语句也包含 else 部分，这样 if 的数目和 else 数目相同，从内层到外层一一对应，不易出错。

（3）若 if 与 else 的数目不一样，有时为了编程的需要，可以加花括号确定配对关系。例如：



这时{ }限定了内嵌 if 语句的范围，所以 else 与第一个 if 配对。程序举例如下：

【例 5.4】编写一程序，把三个整数中最大的打印出来。

```
main( )
    {
    int  a，b，c，m;
    scanf("%d，%d，%d"，&a，&b，&c);
    if(a>b)
       { if(a>c)    m=a; }
    else
       { if(b>c)
            m=b;
         else
            m=c; }
    printf("m=%d"，m);
    }
```

5.3　开关语句switch

switch 语句是多分支选择语句，也叫开关语句。在上一节中介绍了如何用嵌套的 if 结构来解决多路选择问题，我们还可以利用本节将要介绍的开关语句来解决多路选择问题。

5.3.1　switch语句结构形式

开关语句 switch 的一般形式如下：

```
switch(表达式)
   {case  常量表达式 1：语句 1
    case  常量表达式 2：语句 2
                  .
                  .
                  .
    case  常量表达式 n：语句 n
    default            ：语句 n+1
   }
```

例如，根据考试成绩的等级打印出百分制分数段（如上节中所述例子），可以写出下面的程序段:

```
switch(grade)
  {case   '5':    printf("90~100\n");
   case   '4':    printf("75~89 \n");
   case   '3':    printf("60~74 \n");
   case   '2':    printf("0 ~59 \n");
   default   :    printf("error \n");
  }
```

说明：

（1）switch 后面的“表达式”，可以是不同类型的表达式，但最常用的是整型表达式或字符型表达式。

（2）执行 switch 语句，首先计算表达式的值，然后将该值依次与各个情况常量比较，一旦发现某一情况常量与表达式的值相等，则执行相应的 case 后面的语句， 若无相等的情况常量，则执行 default 后面的语句。

（3）在 case 后面若包含了一个以上的执行语句，可以用花括号将其括起来，但也可不加，效果是一样的。

（4）每个 case 的常量表达式的值不能相同。

（5）switch 语句体中可以不包含 default 分支，default 后面只是在找不到匹配的值时才执行，但它的位置并不限定在最后。

（6）执行完一个 case 后面的语句后，程序继续执行下一个 case。“case 常量表达式”只是起语句标号作用，并不是在该处进行条件判断。在执行 switch 语句时，根据 switch 后面的表达式的值找到匹配的入口标号，就从此标号开始执行下去，不再进行判断。

例如，上面程序段中，若 grade 的值等于'5'，则将连续输出：

```
90～100
75～89
60～74
0～59
Error
```

因此，若希望在某一点中止执行 switch 语句，应使用 break 语句(break 语句介绍详见本章第 5.5 节)使流程跳出 switch 结构。

例如，上例程序段可以改写为：

```
switch(grade)
  {case   '5': printf("90~100\n"); break;
   case   '4': printf("75~89\n");  break;
   case   '3': printf("60~74\n");  break;
   case   '2': printf(" 0~59\n");  break;
   default   : printf("error\n");
  }
```

如果 grade 的值为'4'，则只输出“75~89”。流程图见图 5-3 所示。

（7）多个 case 可以共用一组执行语句，如：

```
        .
        .
        .
  case   '5':
```

```
        case   '4':
        case   '3':  printf(">59\n");   break;
                .
                .
                .
```

grade 的值为'5'、'4'或'3'时都执行同一组语句。

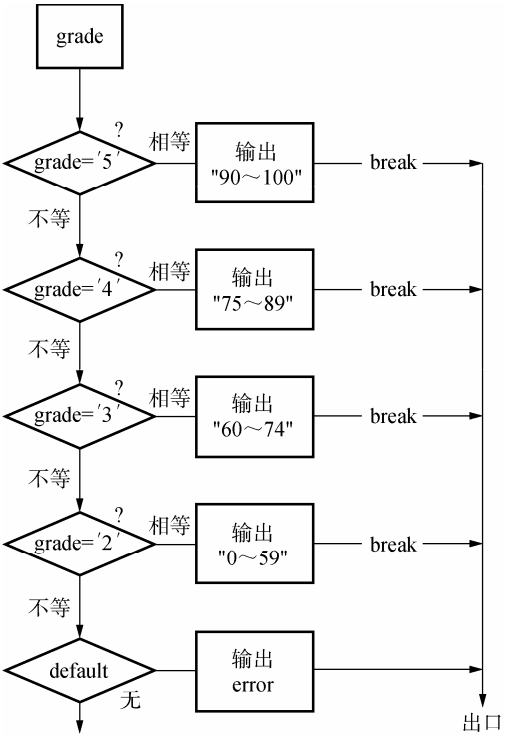


图 5-3　流程图

5.3.2　多路选择程序设计举例

多路选择的程序可以通过 if 语句的第三种形式实现，如【例 5.3】程序。但随着判断条件的增多，程序书写会变的很不方便，写出来可读性也差。用 switch 语句设计多路选择程序，不但方便而且可读性也好。

【例 5.5】编一程序，打印某年某月的天数。

我们知道，一年中各个月份的天数不尽相同，特别是 2 月份还与该年是否闰年有关，故采用 switch 语句，程序如下：

```
main(  )
  {
  int   y，m，length；
  scanf("%d   %d"，&y，&m)；
  switch(m){
    case   1：
    case   3：
    case   5：
    case   7：
    case   8：
    case   10：
```

```
    case  12:   length=31; break;
    case  4:
    case  6:
    case  9:
    case  11:   length=30; break;
    case  2:
      if（(y%4!=0)||(y%100==0&&y%400!=0))
          length=28;
      else  length=29; break;
    default : printf("error, please  again\n"); }
    printf("The length of %d %d is %d\n",y,m,length);
  }
```

【例 5.6】要求程序运行时在屏幕上显示出下面的菜单:

```
1. 加法练习
2. 减法练习
3. 乘法练习
4. 除法练习
5. 退      出
```

如果按“1”键，则显示出一道加法练习题，并判断输入的运算结果是否正确；如果按“2”键，则显示出一道减法练习题……；以此类推，按“5”键时，结束程序的运行。

菜单功能是一种典型的多分支选择，因此，利用 switch 语句来实现菜单选择功能。源程序清单如下：

```
#include  <stdio.h>
main( )
{
 char choice;
 int num;
 printf("\n\n\n");                    /*在菜单上面显示三行空行*/
 printf("\t\t1. 加法练习\n");    /*显示菜单*/
 printf("\t\t2. 减法练习\n");
 printf("\t\t3. 乘法练习\n");
 printf("\t\t4. 除法练习\n");
 printf("\t\t5. 退      出\n");
 printf("\n 请选择(12345)");
 choice=getch();                          /*输入选择项*/
 switch(choice)
  {case'1':printf("\n\n10+27=");
           scanf("%d",&num);
           if(num==10+27) puts("正确!");
           else puts("错误!");
           break;
   case'2':printf("\n\n63-20=");
           scanf("%d",&num);
           if(num==63-20) puts("正确!");
           else puts("错误!");
           break;
   case'3':printf("\n\n2*16=");
```

```
            scanf("%d",&num);
            if(num==2*16) puts("正确!");
            else puts("错误!");
            break;
    case'4':printf("\n\n36/12=");
            scanf("%d",&num);
            if(num==36/12) puts("正确!");
            else puts("错误!");
            break;
    case'5':break;
    default:printf("选择出错! ");
  }
}
```

5.4　循环语句

循环语句是算法语言中应用最普遍也是最重要的语句。C 语言一共提供了三种循环语句，它们可以构成循环控制结构，用来完成在满足一定的条件下需要重复进行操作的任务。c 语言的循环语句虽然和其他高级语言循环语句有很多相似之处，但在书写上更加显示了它的简洁和多变的特点。在进行循环结构程序设计时，要注意灵活对待。

5.4.1　for语句

1. 一般形式

for 语句的一般形式为：

for(表达式 1；表达式 2；表达式 3)　语句

其中，三个表达式要用分号隔开，语句是循环体，可为复合语句。它的执行过程如下：

for 语句是首先求表达式 1 的值；其次判断表达式 2，不为 0（为真）则执行(　)后的语句；此后求表达式 3 的值，再返回来判断表达式 2，不为 0，则再次执行(　)后的语句。这样一直重复执行到表达式 2 的值为 0 为止，不再重复操作，而去执行 for 下面的语句。因此，若第 1 次所求表达式 2 的值就是 0，则 for 语句便只执行表达式 1 就停止了。其流程图见图 5-4 所示。

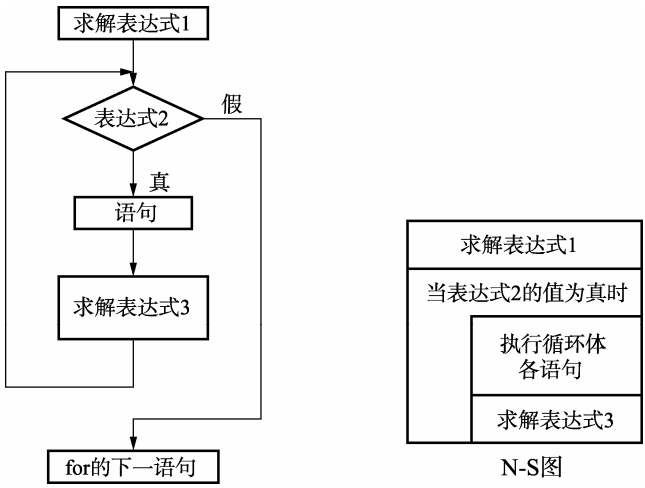


图 5-4　for 循环流程图

说明：

（1）表达式 2 是循环的控制键，一般含有循环控制变量，而表达式 3 通常用来改变循环变量的值。例如：

```
for(m=1；m<=50；m++)  s=s+m；
```

其中 m 是循环变量，当 m 小于等于 50 时，执行循环体语句 s=s+m；表达式 3 m++每执行一次 s=s+m 就加 1，直到 m 大于 50 而结束循环。

（2）表达式 1 可以为循环变量赋初值,也可以与循环变量无关。如:

```
for(s=0；m<=50；m++)  s=s+m；
```

（3）表达式 1、表达式 3 可以是一个简单表达式，也可以是中间用逗号间隔的多个简单表达式，即逗号表达式。如：

```
for(m=0,n=50；m<=n；m++，j--)  k=m+n；
```

逗号表达式按自左至右顺序求解，整个逗号表达式的值为其中最右边表达式的值。如：

```
for(m=1；m<=50；m++，m++)  s=s+m；
```

其中表达式 3 的值,相当于 m=m+2。

（5）表达式 2 一般是关系表达式或逻辑表达式，但也可以是数字表达式或字符表达式，只要其值为非零，就执行循环体。如：

```
for(m=0；(c=getchar())!='\n'；m+=c)；
```

（6）下面给出 for 语句最易理解的形式供读者参考。

For（循环变量赋初值；循环结束条件；循环变量增值） 语句

2. 表达式中的特殊情况

（1）for 语句一般形式中“表达式 1”可以省略，此时应在 for 语句前给循环变量赋初值，但其后的分号不能省略。如：

```
int s=0,m=0;
      :
      :
for(；   m<=50；m++)  s=s+m；
```

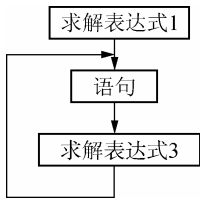


图 5-5　无限制地进行循环

（2）如省略表达式 2， 则不判断循环条件，循环无限制地进行下去。此时表过式 2 始终作为真(见图 5-5)。如：

```
for(m=1；  ；m++)  s=s+m；
```

（3）若省略了表达式 3，此时循环无法结束，应另外设法保证循环能正常结束。如：

```
for(s=o,m=1；m<=50)
    {s=s+m；m++；}
```

（4）表达式 1 和表达式 3 可以同时省略。如：

```
for(；m<=50；)
  {s=s+m；  m++；}
```

（5）三个表达式可以都省略。如：

```
for(;;  ) 语句
```

既不设初值和判断条件，循环变量也不增值，此时认为表达式 2 为真，将无终止地执行循环体。

3. 多重循环

在一个循环体内又包含另一个循环体，称为循环嵌套。内嵌的循环体还可以再嵌套循环，这就是多重循环。例如：

```
for(i=1; i<=9;i++)
   for(j=1;j<=i;j++)
    printf("%d   ",i*j);
```

这是用 for 语句构成的双重循环，用它可以输出三角形 99 乘法表，写出程序如【例 5.7】。后面还要讲到 while 循环和 do—while 循环，这三种循环既可以自身构成嵌套，也可以互相嵌套。

【例 5.7】显示输出三角形 99 乘法表程序。

该乘法表要列出 1×1，2×1，2×2，3×1，3×2，3×3，……，9×9 的值，用外层循环控制乘数，i=1～9，输出 9 行；内层循环控制被乘数，使 j 从 1～i，每行输出 j 列。

```
#include   <stdio.h>
main( )
{
   int i,j;
   for(i=1; i<=9;i++)
      for(j=1;j<=i;j++)
      { printf("%d   ",i*j);
        if(j==i) printf("\n"); }     \*每行输出完最后一列后要换行*\
}
```

运行结果：

```
1
2   4
3   6   9
4   8   12   16
5   10   15   20   25
6   12   18   24   30   36
7   14   21   28   35   42   49
8   16   24   32   40   48   56   64
9   18   27   36   45   54   63   72   81
```

4. for循环程序设计举例

【例 5.8】用循环控制分行输出 count=10、count=20、count=30。

```
#include   <stdio.h>
main(   )
{
     int   count;
     for(count=1；count<=3；count++)
     printf("count=%d\n"，count*10)；
}
```

此例中的循环条件是 count<=3，而循环体是单个语句 printf 函数调用。执行时循环初始 count=1，满足表达式 2 的循环控制条件，打印出 count=10，然后执行表达式 3，使 count=2，再判断满足表达式 2 否，由于 count 的值不断加 1，循环控制条件 count<=3 最终将不能满足，使循环语句终止，整个程序也就结束了。循环条件不断变化这一点非常重要，否则变成死循环。这个程序运行结果为：

```
count=10
count=20
count=30
```

【例 5.9】用近似公式 e=1+1/1!+1/2!+…+1/n!求自然对数的底 e 的值，取 n 为 10(n 愈大愈接

近 e 的真值)，用一层循环实现的程序如下：

```
#include   <stdio.h>
main()
 {
    int   n；
    float   e，p；
    e=p=1.0；
    for(n=1；n<=10；n++)
   { p*=n；
    e=e+1.0/p; }
     printf("e=%10.7f\n"，e)；
 }
```

运行结果：

```
      e=2.7182810
```

请读者课后用两层循环实现该程序。

【例 5.10】求 3~100 之间的所有素数，按每行 4 列输出。

所谓素数，是指除了能被 1 和自身整除外，不能被任何其他整数整除的自然数，如 3、5、7、11 等。验证一个自然数 m 是否素数，有很多办法，最容易理解的方法是：将 m 除以 2～(m–1) 的每一个自然数，如果都除不尽，则 m 是素数；只要有一个能除尽，m 就不是素数。

根据此算法思路，该程序应该用双重循环实现。外循环用来控制被验证的数在 3～100 之间，内循环用来验证某个数 m 是否为素数，并按每行 4 列输出。

源程序设计如下：

```
#include   <stdio.h>
main( )
{
 int   i,m, k,j=0;
 for(m=3;m<=100;m++)          /*控制被验证的数在 3~100 之间*/
   { k=0;                     /*用 k=0 作为素数的标志*/
     for( i=2;i<=m-1;i++)     /*验证某个数 m 是否为素数*/
       if (m%i==0)
       {k=1;break;}           /*用 k=1 标志某数不是素数*/
     if (k==0)
       { printf("%3d",m);
         j=j+1;
         if (j%4==0)          /*用来控制每行输出 4 个素数*/
         printf("\n");
       }
   }
}
```

运行结果如下：

```
  3   5   7  11
 13  17  19  23
 29  31  37  41
 43  47  53  59
 61  67  71  73
 79  83  89  97
```

5.4.2　while语句

While 语句用来构建“当型”的循环结构。当循环判断条件为真时，就执行循环体内语句。其应用格式及说明如下：

1. 一般格式

while(表达式)　语句

说明：

（1）其中表达式是循环能否继续重复的判断条件；循环体(即语句部分)可以是单一语句或复合语句。

（2）执行过程是：首先计算表达式，若其值非 0，则执行循环体并重新计算表达式，循环一直继续到表达式的值变成 0 为止。其流程图见图 5-6 所示。

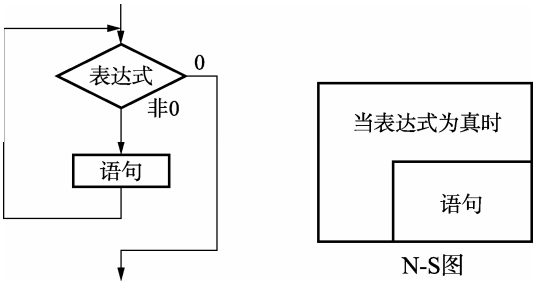


图 5-6　while 循环流程图

利用 while 循环可以将【例 5.8】改写为如下程序：

【例 5.11】“当型”的循环结构程序

```
#include   <stdio.h>
main( )
{
int   count=1;
while(count<=3)
printf("count=%d\n"，count++*10);
}
```

注意

（1）循环体若是复合语句，需用花括号括起来。不然，while 的作用范围只到其后的第一个分号。

（2）在循环体中应有使循环趋向于结束的语句。

（3）while 语句和 for 语句相比，没有 for 语句功能强，for 语句不仅可以用于循环次数已确定的情况，而且可以用于循环次数不确定而只给出循环结束条件的情况，它完全可以代替 while 语句。for 语句的一般形式可以改写如下形式：

```
表达式 1;
while(表达式 2)
{语句
表达式 3;
}
```

2. while循环程序举例

【例 5.12】令 i 的初值为 0，测试 i 是否小于 5；若为真，则显示 i 的值；然后 i 值递增 1， 再进行测试，直到 i<5 不成立，显示：停止循环。程序如下：

```
#include   <stdio.h>
main(   )
        {
          int   i=0;
          while(i<5)
          printf("%d\n，i++);
          printf("停止循环。\n");
        }
```

执行结果：

```
0
1
2
3
4
停止循环。
```

【例 5.13】华氏温度 F 与摄氏温度 C 的关系是：C=5/9*(F–32)，如果已知 F 为–10，20，50，……，200，用 C 程序求出各华氏温度所对应的摄氏温度。程序如下：

```
#include   <stdio.h>
main( )
  {
   float   fa，ce;
   fa =-10;
   while(fa<=200)
    { ce=(5.0/9.0)*(fa-32.0);
      printf("%4.0f      %6.1f\n"，fa，ce);
      fa=fa+30；}
  }
```

执行结果：

```
-10          -23.4
 20           -6.7
 50           10.0
 80           26.7
110           43.3
140           60.0
170           76.7
200           93.3
```

【例 5.14】从键盘依次输入学生的成绩，并进行计数、累加，当输入–1 时，停止输入，进行输出学生的数量、总成绩和平均成绩。

这是一个“终止标志使用”的例题，在处理实际问题时经常会遇到这样的问题。我们设–1 为输入学生成绩的结束标志，n 为学生的计数变量，t 为总成绩的累加变量，x 为每个学生成绩的暂存变量。源程序设计如下：

```
#include   <stdio.h>
main( )
  {
    int   n=0;
    float t=0,x;
    scanf("%f",&x);
    while(x!=-1)
      {
        n+=1;
        t+=x;
        scanf("%f",&x); }
     printf("n=%d t=%6.2f pj=%6.2f\n",n,t,  t/n);
  }
```

程序运行时，若从键盘键入如下内容：

```
100   70   80   85   75 -1↙
```

输出结果：

```
n=5   t=410.00   pj=82.00
```

5.4.3　do-while语句

do-while 语句用来构建“直到型”的循环结构。这种结构先执行循环体内语句，直到循环判断条件为假时，就结束循环。其应用格式如下：

1. 一般格式

do-while 语句的一般格式为：

```
   do
    { 循环体语句 }
   while  (表达式);
```

其执行过程是：先执行循环体语句部分，然后判别表达式，当表达式为真时（非 0），返回重新执行循环体语句部分，如此反复，直到表达式的值为假（0 值），此时循环结束，执行循环后面的语句序列。流程图见图 5-7 所示。

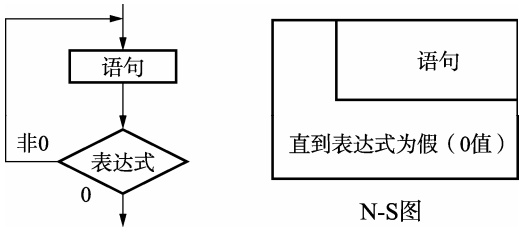


图 5-7　do-while 循环流程图

若将【例 5.12】改为使用 do-while 语句，程序如下：

【例 5.15】“直到型”的循环结构程序

```
#include   <stdio.h>
main(   )
  {
     int   i=0;
     do{
```

```
            printf("%d\n"，i++);
          }
      while(i<5);
      printf("停止循环\n");
   }
```

2. -while与while比较

用 while 语句和 do-while 语句编写程序，所得结果有些是一样的，如【例 5.12】和【例 5.15】就可分别用两种语句编程，其结果一样。有些则不相同，见下面例题：

求两个非 0 整数之和。分别设计程序如下：

【例 5.16】使用 while 语句

```
#include  <stdio.h>
main(  )
  {
    int  x，y，z;
    scanf("%d,%d"，&x，&y);
    while(x!=0&&y!=0)
      { z=x+y;
        printf("z=%d\n"，z);
        scanf("%d,%d"，&x，&y);
      }
  }
```

运行结果：

```
4，5
z=9
2，3
z=5
0，0
```

【例 5.17】使用 do-while 语句

```
#include  <stdio.h>
main( )
  {
    int  x，y，z;
    do{
       scanf("%d,%d"，&x，&y);
       z=x+y;
       printf("z=%d\n"，z);
      }
    while(x!=0&&y!=0);
  }
```

运行结果：

```
4，5
z=9
2，3
z=5
0，0
z=0
```

由以上二个程序可以看出：

（1）do-while 语句至少要执行一次循环体。而 while 语句先判断(　)内的值，若为 0，则跳出 while 循环，因此可能循环体一次也不执行。 当 x 和 y 全部输入 0 时，【例 5.16】不执行 x+y 和 printf()两个语句跳出循环， 而【例 5.17】则执行了上述两个语句跳出循坏，它在屏幕终端上显示出 z=0 后，再终止执行。

（2）两个语句中的循环体基本相同。

（3）这两种循环的循环变量初始化一般都在循环体之前进行，但 do-while 结构有时也可以在循环体内进行（如【例 5.17】）。

请看下面两个例题：要求设计求 100 以内（含 100）偶数之和的 C 程序（循环变量初始化在循环体之前进行）。

【例 5.18】用 while 语句实现

```
#include   <stdio.h>
main( )
  {
    int   x,y;x=y=0;      /*循环变量 x  初始化为 0*/
    while(x<100);
     {
       ++x;++x;
       y+=x;
      }
     printf("2+4+6+·····+98+100=%d\n",y);
  }
```

【例 5.19】用 do-while 语句实现

```
#include   <stdio.h>
main( )
  {
    int   x,y;x=y=0;      /*循环变量 x  初始化为 0*/
    do{
         ++x;++x;
         y+=x;
        }
     while(x<100);
     printf("2+4+6+·····+98+100=%d\n",y);
  }
```

注意

以上学习的三种基本循环结构，在一般情况下是可以互相替代的，但使用时会有一些差别，要注意区别对待。for 循环功能强、使用灵活，是人们使用最普遍的。在解决具体问题时究竟使用哪一种结构，要看使用者的习惯、风格和写出程序的可读性、可维护性等。

5.5　中止语句break和继续语句continue

5.5.1　break语句

break 语句，像在 switch 语句中那样，提供了在 while、for、do-while 语句中的早期出口，

即 break 语句导致从包含它的循环语句或开关语句中立即跳出。这种不在顶部(while 和 for)或底部(do-while)通过判断就能控制循环出口的做法是非常方便的。中止语句的结构是：

```
break;
```

【例 5.20】利用 break 跳转的程序

```
#include  <stdio.h>
main( )
  {
    int  i;
    for(i=1; i<6; i++){          /*i 从 1 到 5 的循环*/
        if(i==3)  break; }      /*当 i=3 时跳出循环*/
    printf("In the break loop,i is now %d\n",i);
  }
```

运行结果：

```
In the break loop,i is now 3
```

例中虽然 for 循环语句规定 i 从 1 到 5 循环，但由于循环体中有“if() break;”语句，所以实际上当循环到 i=3 时就跳出循环。通常，break 语句总是和 if 语句联系在一起，用以判断在满足什么条件下才跳出循环。如果是多重循环，只是跳出所在的循环体，可能还在外层的循环体中。

5.5.2 continue语句

continue 语句的一般形式是：

```
continue;
```

continue 语句只能用在 while、for 或 do-while 的循环体中，不适用于 switch 语句。一般 continue 也是跟条件语句一起使用的，遇到 continue 则跳过循环体中下面的语句而从下一次循环开始。

【例 5.21】利用 continue 跳转的程序

```
#include  <stdio.h>
main( )
  {
    int  i;
    for(i=1; i<6; i++){    /*i 从 1 到 5 的循环*/
     if(i==3) continue;    /*i=3 时跳过显示语句*/
     printf("In the continue loop,i is now %d\n",i);}
  }
```

运行结果：

```
In the continue loop,i is now 1
In the continue loop,i is now 2
In the continue loop,i is now 4
In the continue loop,i is now 5
```

请读者自己分析 break 语句与 continue 语句的类似之处和本质上的区别。

5.6 无条件转移语句goto

5.6.1 goto语句的格式

goto 语句为无条件转向语句，它的一般形式为：

goto　语句标号；
……
语句标号：……；
……

语句标号可放在任何语句的开始处，标号的后面要加冒号。语句标号用标识符表示，它的定名规则与变量名相同，不能用整数来作标号，例如：

goto　abc_c；　　是合法的
goto　52；　　是非法的

结构化程序设计方法主张限制使用 goto 语句。如果过多地使用，会使得程序的执行情况变得错综复杂、可读性差。但也不是绝对禁止，一般来说，可以有两种用途：

（1）与 if 语句一起构成循环结构。

（2）从循环体中跳到循环体外。但在 c 语言中，由于可以用 break 语句和 continue 语句跳出本层循环和结束本次循环。因此，使 goto 语句的使用机会大大减少。并且不能用 goto 语句直接进入循环体。

在程序设计时，既使要使用 goto 语句，也要有控制地使用，尽可能少用。

5.6.2　goto语句应用举例

【例 5.22】用 goto 语句构成循环求 1 到 100 的整数和。

```
#include   <stdio.h>
main( )
{
   int i=1,sum=0;
   loop: sum=sum+i;
         i++;
         if(i<=100) goto loop;
         else
         printf("1+2+3+…+100=%d",sum);
}
```

程序运行结果：

```
    1+2+3+…+100=5050
```



习题五

1. 单项选择题

（1）C 语言用_____表示逻辑量为“假”。

A. FALSE　　B. F　　C. 数值 0　　D. 非零值

（2）要求当 A 的值为奇数时，表达式的值为“真”；A 的值为偶数时，表达式的值为“假”。以下不满足要求的表达式是_____。

A. A%2==1　　B. !(A%2==0)　　C. A%2　　D. !(A%2)

（3）在 if 语句的嵌套中，else 总是与_____配对。

A. 它前面未配对的 if　　B. 它上面书写在同一列的 if

C. 它前面最近的未配对的 if　　D. 它在同一行的 if

（4）下面对 for 循环概念描述正确的是_____。

A. 只能用于循环次数已确定的情况　　B. 不能用 break 语句跳出循环体

C. 先执行循环语句后判定表达式　　D. 循环体中可包含用{ }括起来的多条语句

2. 填空题

（1）switch 后面的“表达式”，通常为_____型或_____型表达式；每个 case 的常量表达式的值_____。

（2）如果不算 goto 语句构成的循环结构，那么 C 语言可以构建循环的语句一共有____种,分别是_____。

（3）continue 语句与 break 语句在使用时，有两点主要的不同：①它不适用于_____语句；②它跳转时是_____。

（4）把下面的两条 if 分支语句合并成一条 if 分支语句应该写成_____。

```
if(x<y) n=1;
else m=2;
if(x>=y)   printf(”m=%d\n”,m);
else   printf(”n=%d\n”,n);
```

3. 指出以下程序的打印结果

```
main(  )
  {
   int  x，y，z;
   x=y=1;
   while(y<10)
   ++y;
   x+=y;
   printf("x=%d  y=%d\n"，x，y);
   x=y=1;
   while(y<10)
   x+=++y;
   printf("x=%d  y=%d\n"，x，y);
   y=2;
   while(y<10){
   x=y++;    z=++y; }
   printf("x=%d  y=%d  z=%d\n"，x，y，z);
   for(y=2；(x=y)<10；y++)
   printf("x=%d  y=%d\n"，x，y);
   for(y=2；(x=y)<10  y++)
   printf("x=%d  y=%d\n，x，y);
   fot(x=1，y=1000；y>2；x++，y/=10)
   printf("x=%d  y=%d\n"，x，y);
  }
```

4. 编写程序

（1）用两种循环语句编写以下各个计算公式的程序：

① 计算球体体积　$v=4/3\pi R^3$

初值 R=1，终值 R=100，步长为 2。

② s=13+23+33+…+103

（2）某一次考试之后，需要找出其中的最高分和最低分。试为此编一程序。

（3）打印出所有的“水仙花数”，所谓“水仙花数”是指一个三位数，其各位数字立方和等于该数本身。



上机实习指导

一、学习目标

本章重点介绍了控制语句中的分支语句、循环语句以及相关的辅助语句和表达式。这些控制语句是学习较复杂程序设计的基础，而相关的关系表达式和逻辑表达式又是学好分支结构和循环结构的基础。因此，本章的内容十分重要，所需学时和其他各章相比也是最多的。通过本章学习，应达到以下要求：

1. 熟练掌握关系运算符和逻辑运算符的用法，能准确判断关系表达式和逻辑表达式的值，并能自如应用于分支与循环程序中的条件判断；

2. 理解程序设计中的几种常用算法的基本思想。

3. 掌握控制结构程序的设计的方法；

二、应注意的问题

1. 关于表达式的值

关系表达式和逻辑表达式的值只有两种：1 和 0，即表达式为真时其值为 1，表达式为假时其值为 0。而在判断真与假时，则 0 为假，非 0 为真。

要注意，在 C 语言中表示一个变量的值在某两个数之间时，和数学中的表达完全不同，不要用错。如，要表示 x 的值在 3 和 10 之间，绝不能写成：3<x<10，而应写成 3<x && x<10。

2. 关于程序的书写格式

程序的书写格式好坏，关系到程序的可读性和可维护性，而这也是结构化程序设计方法强调的要点之一。要使程序清晰易读，一个较好的做法是采用缩格书写形式(请见书中的各例题)，这对结构、段落、嵌套能一目了然。对初学者来说，无论是在纸上写程序，还是上机建立源程序，都应注意程序的书写格式，用以提高调试程序的效率。

3. 关于 switch 语句的适用范围

switch 语句能够很方便地实现多分支选择，但是，使用 switch 语句只能判断一个表达式的值与一个常量是否相等关系，而不能直接判断大于、小于等复杂的关系。



上机实习一　关系表达式和逻辑表达式

一、目的要求

1. 掌握各种关系运算符和逻辑运算符的功能；

2. 能正确理解“真”、“假”的概念，以及其在 C 语言中表示形式；

3. 能够把一个命题写成符合 C 语言语法规则的关系表达式或逻辑表达式。

二、上机内容

1. 运行下列程序，分析并观察运行结果

（1）main()

```
{
```

```
 int a,b,c;
 a=1;b=2;c=3;
 printf("a=%d,b=%d,c=%d\n",a,b,c);
 printf("a<b 的值为：%d\n",a<b);
 printf("a<=b 的值为：%d\n",a<=b);
 printf("a==c 的值为：%d\n",a==c);
 printf("a!=c 的值为：%d\n",a!=c);
 printf("c>b 的值为：%d\n",c>b);
}
```

（2）main()

```
{
 int x=3,y=4,z=5,b;
 printf("x=%d,y=%d,z=%d\n",x,y,z);
 b=x<y&&x<z;          printf("x<y&&x<z 的值为：%d\n",b);
 b=x<y&&x>z;          printf("x<y&&x>z 的值为：%d\n",b);
 b=x<y||x>z;          printf("x<y||x>z 的值为：%d\n",b);
 b=!x;                printf("!x 的值为：%d\n",b);
 b=!(x>y);            printf("!(x>y 的值为：%d\n",b);
 b=(x>y||x>z)&&y<z;printf("x>y||x>z)&&y<z 的值为：%d\n",b);
}
```

（3）main()

```
{
 int x=20,y;
 y=2<x<10;
 printf("x=%d\n",x);
 printf("2<x<10 的值为：%d\n",y);
 y=2<x&&x<10;
 printf("2<x&&x<10 的值为：%d\n”,y);
}
```

2. 完善程序

已知，闰年符合下面两个条件之一：

（1）能被 4 整除，但不能被 100 整除；

（2）能被 400 整除。

下面的程序验证 2000 年为闰年，请在横线处填写正确的运算符或表达式，使程序完整。程序的后面给出了运行结果。



提示：可以通过求余运算判断能否整除。若 a%b 的值为 0，则表明 a 能被 b 整除。

```
main( )
{
 int year=2000,b;
 b=(year%4==0___year%100___0)___(_________________);
 printf(”%d\n”,b);
}
```

运行结果：

```
1
```



上机实习二　if语句

一、目的要求

1. 进一步熟悉关系表达式和逻辑表达式；
2. 熟练掌握 if 语句的三种形式，能阅读、分析和设计分支程序。

二、上机内容

1. 运行下列程序，分析并观察运行结果

（1）main()

```
{
 int number;
 printf("number=");
 scanf("%d",&number);
 if (number%10==0) printf("%d 是 10 的倍数。",number);
}
```

说明：

运行这个程序时，分别输入下面两个测试数据，注意观察各自的运行结果。

测试数据一：

```
300 ↙
```

测试数据二：

```
27 ↙
```

（2）main()

```
{
 int number;
 printf("number=");
 scanf("%d",&number);
 if (number%10==0) printf("%d 是 10 的倍数。",number);
 else printf("%d 不是 10 的倍数。",number);
}
```

说明：

该程序的测试数据同程序（1），注意运行结果与（1）有什么不同。

（3）main()

```
{
 int number;
 printf("number=");
 scanf("%d",&number);
 if (number>0) printf("%d 是正数。",number);
 else if (number==0) printf("%d 是零。",number);
 else printf("%d 是负数。",number);
}
```

2. 完善程序

请在横线处填写正确的表达式或语句，使程序完整。上机调试程序，使程序的运行结果与

书中的结果一致。

```
main( )
{
  int m,days;
  printf(”输入月份：”); scanf(”%d”,&m);
  if (_____________________________)
      days=31;
  else if (m==4||m==6||m==9||m==11)
      _________;
  else days=29;
  printf(”_____________________”,m,days);
}
```

运行结果一：

```
输入月份：1 ↙
1 月份有 31 天。
```

运行结果二：

```
输入月份：9 ↙
9 月份有 30 天。
```

运行结果三：

```
输入月份：2 ↙
2 月份有 29 天。
```



上机实习三　switch语句

一、目的要求

1. 掌握 switch 语句的功能、使用格式和执行过程；
2. 能用 switch 语句设计多分支程序。

二、上机内容

1. 运行下列程序，分析并观察运行结果

对下面程序，要求自行设计几组有代表性的输入数据，这些数据要能分别覆盖程序中的各条分支。

```
main( )
  {
    int n;
    scanf("%d",&n);
    switch(n)
      {case 1:puts("*");break;
       case 2:puts("**");break;
       case 3:puts("***");break;
      }
  }
```

2. 完善程序

下面程序的功能是，输入一个百分制成绩，输出用 A、B、C、D、E 表示成绩的等级。已

知 90 分以上为 A，80 至 89 分为 B，70 至 79 分为 C，60 至 69 分为 D，60 分以下为 E。

在横线处填写正确语句或表达式，使程序完整，并调试程序，使程序的运行结果与给出的结果一致。

```
main ( )
{
  int grade;
  printf("输入成绩：");scanf("%d",&grade);
  grade=grade/10;
  switch(_________)
   {
    case 10:
    case 9:printf(_____________);break;
    case 8:printf("等级为 B");_______;
    case 7:_____________________;_________;
    case 6:_____________________;_________;
    default:____________________;_________;
   }
}
```

运行结果一：

```
输入成绩：98 ↙
等级为 A
```

运行结果二：

```
输入成绩：100 ↙
等级为 A
```

运行结果三：

```
输入成绩：80 ↙
等级为 B
```

运行结果四：

```
输入成绩：65 ↙
等级为 D
```

运行结果五：

```
输入成绩：51 ↙
等级为 E
```

3. 编写程序

编一程序，要求实现下面的功能：

输入一个实数后，屏幕上显示如下菜单：

```
1. 输出相反数
2. 输出平方数
3. 输出平方根
4. 退　　出
```

若按 1 键，则输出该数的相反数；若按 2 键，则输出该数的平方数；……。按 1～4 之外的其他键时，显示出错。



提示：为了输出界面整洁美观，可在程序输入、输出之前，加上一个清屏幕的函数调用，该函数为 clrscr()。

求数 x 平方根的标准函数为 sqrt(x)。常用数学标准函数的调用请见附录 B 第一部分，使用时要加上头文件 math.h。



上机实习四　for语句

一、目的要求

1. 掌握 for 语句的功能、使用格式和执行过程；

2. 理解 for 语句主要用于已知循环次数的循环结构，初步掌握用 for 语句设计循环程序的方法。

二、上机内容

1. 运行程序，分析并观察运行结果

（1）main()

```
{
 int i;
 clrscr();
 for(i=1;i<=20;i++)
    if(i%4==0)printf("%d    ",i);
}
```

（2）main()

```
{
 int i,j;
 clrscr();
 for(i=1;i<=4;i++)
    {for(j=1;j<=i;j++)
       printf("*");
     printf("\n");
    }
}
```

说明：

将 for(i=1;i<=4;i++)改成 for(i=1;i<=10;i++)后，再运行程序，注意观察运行结果的改变。

2. 完善程序

下面程序的功能是打印如下图形，请在横线处填写正确语句或表达式，使程序完整，上机调试程序，使程序的运行结果与给出的结果一致。

```
        *
       * *
      * * *
     * * * *
main( )
 {
  int i,j;
  clrscr();
  for(i=0;i<=4; ++i)
    { for(j=0;j<_______;j++)
```

```
            printf("    ");
        for(j=0;j<_________;j++)
            printf(" *");
            printf("\n");
        }
}
```

3. 编写程序

输入 10 个学生成绩，要求统计出及格人数和不及格人数。



上机实习五　while语句和do-while语句

一、目的要求

1. 掌握 while 语句和 do-while 语句的功能、使用格式和执行过程；

2. 了解 while 语句与 do-while 语句的不同之处。

3. 理解 while 语句和 do-while 语句主要用于已知循环条件的循环结构，初步掌握用其设计循环程序的方法。

二、上机内容

1. 运行下列程序，分析并观察运行结果

（1）main()

```
{
 int i,sum;
 clrscr();
 i=1;
 sum=0;
 while(i<=3)
    {sum+=i;
       i++;
    }
 printf("sum=%d",sum);
}
```

说明：

将程序中的 i=1 改成 i=4 后，再运行程序，注意观察运行结果有什么变化。

（2）main()

```
{
 int i,sum;
 clrscr();
 i=1;
 sum=0;
 do
    {sum+=i;
       i++;
    }
 while(i<=3)
```

```
        printf("sum=%d",sum);
    }
```

说明：

将程序中的 i=1 改成 i=4 后，再运行程序，注意观察运行结果有什么变化。

2. 完善程序

下面程序要求：输入一组学生成绩，统计出及格人数和不及格人数。当输入成绩为-1 时结束输入。根据程序的功能，在横线处填写正确语句或表达式，使程序完整。上机调试程序，使程序的运行结果与给出的结果一致。

```
main( )
{
 int grade,n1,n2     /*n1 存放及格人数，n2 存放不及格人数*/
 clrscr();
 n1=n2=____;
 printf("输入一组学生成绩：\n");
 _____________________________;
 while(_____________________)
   {if(grade>=60) n1++;
    else _________;
    scanf("%d",&grade);
   }
 printf("及格人数为：%d\n 不及格人数为：%d",n1,n2);
}
```

运行结果：

```
输入一组学生成绩：
80 ↙
74 ↙
51 ↙
96 ↙
48 ↙
-1 ↙
及格人数为：3
不及格人数为：2
```

3. 编写程序

编写程序，打印出 10 行杨辉三角形：

```
        1
        1 1
        1 2 1
        1 3 3 1
        1 4 6 4 1
        1 5 10 10 5 1
        ……
```

请上机前进行预习，想好算法及实现的步骤，最好先在纸上写好源程序，再上机调试通过。

第6章　数　　组

数组是几乎每一种高级语言都提供了的数据类型，它可以有序地存放一组相关的并且具有相同类型的数据。在程序设计中，对处理大量、相关、有序的数据时，使用数组会在定义、管理和操作方面带来极大的方便。

【本章要点】

1. 数值型数组的定义和引用；
2. 字符型数组的定义和引用；
3. 字符串的处理。

【学习目标】

1. 了解数组的概念，掌握数组的定义方法；
2. 掌握数组的初始化，能正确地引用数组；
3. 了解字符数组与字符串的区别和联系，能够运用字符数组存储和处理字符串；
4. 在实际编程中能够灵活地运用数组来解决实际问题。

【课时建议】

讲授 6 课时，上机 4 课时（利用机动课时 2）。

6.1　一维数组的定义和引用

6.1.1　一维数组的定义

数组是有序的并具有相同类型的数据的集合。例如，一组学生的成绩、一串文字等都可以用数组来表示。同一数组中的各个元素具有相同的数组名和不同的下标。

1. 一维数组的定义

与简单变量的使用一样，在使用数组之前必须要先定义数组。定义一维数组的一般形式为：

```
类型说明符　数组名[常量表达式];
```

例如：

```
int num[10];
```

该语句定义了一个名为 num 的整型数组，数组中共有 10 个元素。

说明：

（1）类型说明符：类型说明符定义了数组的类型。数组的类型也是该数组中各个元素的类型，在同一数组中，各个数组元素都具有相同的类型。

（2）数组名：数组名的命名规则与变量名相同，即遵循标识符的命名规则。

（3）常量表达式：数组名后面用方括号括起来的常量表达式，表示数组中元素的个数，即

数组的长度。需要注意的是，常量表达式中可以包含常量或符号常量，但不能包含变量，也就是说，C 语言中不允许对数组的大小作动态定义。

例如，下面这种定义数组的方法是非法的：

```
int n;
scanf ("%d",&n) ;
int a[n] ;
```

（4）如果数组的长度为 n，则数组中第一个元素的下标为 0，最后一个元素的下标为 n−1。

例如，若定义了下面的数组：

```
int num[10];
```

则 num 数组中的 10 个元素分别为 num[0]、num[1]、num[2]、……、num[9]。

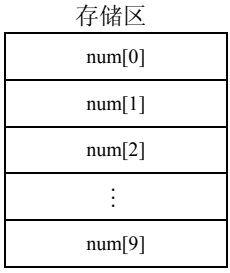
2. 一维数组的机内表示

数组是一组有序的数据，其有序表现在同一数组中各个元素在内存中的存放顺序上。C 语言编译程序分配一片连续的存储单元来存放数组中各个元素的值。

例如：

```
int num[10];
```

num 数组中的各个元素在机内的存储顺序为：



可以看出，下标相邻的数组元素，在机内占有相邻的存储单元。

6.1.2 一维数组的引用

在 C 语言中，使用数值型数组时，只能逐个引用数组元素而不能一次引用整个数组。数组元素的引用是通过下标来实现的。

一维数组中数组元素的表示形式为：

```
数组名[下标]
```

说明：

（1）引用数组元素时，下标可以是任何整型常量、整型变量或任何返回整型量的表达式。例如：

```
num[5]，score[3*9]，
a[n] （n 必须是一个整型变量，并且必须具有确定的值），
num[5]=score[0]+score[1];
```

（2）如果一维数组的长度为 n，则引用该一维数组的元素时，下标的范围为 0～n−1。例如：

```
int num[10];
```

则各个数组元素顺序为：num[0]、num[1]、num[2]、num[3]、……、num[9]，不存在 num[10]元素。

（3）对数组元素可以赋值，数组元素也可以参加各种运算，这与简单变量的使用是一样的。

【例 6.1】一维数组的引用。

```
main()
{
  int i,a[5];
  clrscr();     /* 清屏 */
  for(i=0;i<=4;i++)
     a[i]=i;
  for(i=0;i<=4;i++)
     printf("a[%d]=%d\n",i,i);
}
```

运行结果如下：

```
a[0]=0
a[1]=1
a[2]=2
a[3]=3
a[4]=4
```

程序中的第一个 for 循环使 a[0]到 a[4]的值分别为 0～4，第二个 for 循环则顺序输出 a[0]到 a[4]的值。

【例 6.2】输入 10 个学生的成绩，求这 10 个学生的总成绩和平均成绩。

```
#define N 10
main()
{
  int i,score[N];
  int sum;
  float average;
  clrscr();
  printf("输入%d 个学生的成绩：\n",N);
  for(i=0;i<N;i++)
     scanf("%d",&score[i]);
  sum=0;
  for(i=0;i<N;i++)
     sum+=score[i];
  average=(float)sum/N;
  printf("总成绩为：%d\n",sum);
  printf("平均成绩为：%.2f",average);
}
```

运行结果如下：

```
输入 10 个学生的成绩：
87   85   90   93   76   64   83   91   78   94 ↙
总成绩为：841
平均成绩为：84.10
```

程序中使用了两个 for 循环，分别用于输入 10 个学生的成绩和计算 10 个学生的总成绩。这两个 for 循环也可以合并成一个循环。改进后的程序如下：

```
#define N 10
main()
{
```

```
    int i,score[N];
    int sum;
    float average;
    clrscr();
    printf("输入%d 个学生的成绩：\n",N);
    for(i=0,sum=0;i<N;i++)
       {
  scanf("%d",&score[i]);
  sum+=score[i];
       }
    average=(float)sum/N;
    printf("总成绩为：%d\n",sum);
    printf("平均成绩为：%.2f",average);
  }
```

【例 6.3】分别输入 5 个学生的语文成绩、数学成绩和 C 语言成绩，求每个学生的总成绩和平均成绩。

```
  #define N 5
  main()
  {
    int score1[N],score2[N],score3[N],sum[N];    /* 数组 score1、score2、score3 分别存放三门课程的成绩，数组 sum 存放总成绩 */
    int i;
    clrscr();
    for (i=0;i<N;i++)
      {
       printf("输入第%d 个学生的三门成绩：",i+1);
       scanf("%d%d%d",&score1[i],&score2[i],&score3[i]);    /* 输入三门成绩 */
       sum[i]=score1[i]+score2[i]+score3[i];      /* 计算总成绩 */
      }
    for (i=0;i<N;i++)    /* 输出总成绩和平均成绩 */
      {
       printf("第%d 个学生的总成绩和平均成绩分别为：%d，%.2f\n",i+1,sum[i],sum[i]/3.0);
      }
  }
```

运行结果如下：

```
输入第 1 个学生的三门成绩：83  90  86 ↙
输入第 2 个学生的三门成绩：88  76  90 ↙
输入第 3 个学生的三门成绩：68  72  87 ↙
输入第 4 个学生的三门成绩：75  79  80 ↙
输入第 5 个学生的三门成绩：90  95  92 ↙
第 1 个学生的总成绩和平均成绩分别为：259，86.33
第 2 个学生的总成绩和平均成绩分别为：254，84.67
第 3 个学生的总成绩和平均成绩分别为：227，75.67
第 4 个学生的总成绩和平均成绩分别为：234，78.00
第 5 个学生的总成绩和平均成绩分别为：277，92.33
```

此例分别用了 3 个数组来存放每个学生的三门课程成绩，在后面的 6.2.2 节中，将看到如何使用一个二维数组来存放所有学生的全部成绩数据。

6.1.3　一维数组的初始化

使数组元素具有某个值的方法很多，例如，可以用赋值语句给数组元素赋值，也可以使用输入函数在程序运行时给数组元素赋值（如【例 6.2】和【例 6.3】），除此之外，还可以在定义数组时，对数组元素赋初值，即初始化数组。

（1）定义一维数组时，数组元素的初值依次放在一对花括号内，每个值之间用逗号间隔。例如：

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
```

经过上面的初始化之后，数组元素 a[0]的值为 0，a[1]的值为 1，a[2]的值为 2，……，a[9]的值为 9。

（2）可以只给一部分数组元素赋初值。例如：

```
int a[10]={87,35,12,54,60,58};
```

这里，只给前面的 6 个数组元素（a[0]～a[5]）赋了初值，而后面 4 个没有赋初值的数组元素（a[6]～a[9]），则被自动初始化为 0。

（3）对全部的数组元素赋初值时，可以不指定数组的长度。例如：

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
```

可以写成：

```
int a[ ]={0,1,2,3,4,5,6,7,8,9};
```

这里，由于花括号中有 10 个值，所以系统自动定义数组 a 的长度为 10。但如果希望数组的长度大于提供的初值的个数，那么在定义数组时，方括号内的长度值不能省略。

【例 6.4】逆序输出数组中各元素的值。

```
main()
{
 int a[5]={10,13,2,16,3};
 int i;
 clrscr();
 printf("数组 a 中各元素的值分别为：");
 for(i=0;i<5;i++)
    printf("%5d",a[i]);
 printf("\n");
 printf("逆序输出的结果为：");
 for(i=4;i>=0;i--)
    printf("%5d",a[i]);
 printf("\n");
}
```

运行结果如下：

```
数组 a 中各元素的值分别为：10   13   2   16   3
逆序输出的结果为：3   16   2   13   10
```

6.1.4　一维数组应用举例

【例 6.5】输入一组整数，输出其中的最大值。

```
main()
{
 int num[10],i;
```

```
    int max;
    clrscr();
    printf("输入 10 个整数：\n");
    for(i=0;i<10;i++)
       scanf("%d",&num[i]);
    max= num[0];
    for (i=1 ; i<10 ; i++)      /*  求最大值  */
       if (num[i]>max) max=num[i];
  printf("最大值  = %d\n",max);
  }
```

运行结果如下：

```
输入 10 个整数：
18   32   56   −28   0   10   188   −51   23   1 ↙
最大值=188
```

程序分析：

上面的程序中，使用数组 num 存放输入的 10 个整数，变量 max 存放这组数中的最大值。首先，用 num 数组中的第一个元素（即 num[0]元素的值）来初始化 max 变量，再通过循环语句，依次把 num[1]～num[9]的值，与 max 相比较，如果数组元素的值比 max 的值大，则把该元素的值赋给 max。使用类似的方法，还可以求一组数中的最小值。

【例 6.6】用选择法对 10 个数排序（按从小到大的顺序）。

程序分析：

选择排序法的排序过程是：使用数组存放要排序的一组数（假设有 n 个数），首先从 n 个数中找出最小值，将它放在数组的第一个元素位置上，再在剩下的 n−1 个数中找出最小值，放在第二个元素位置上，……，这样不断重复下去，直到只剩下最后一个数为止。

例如：

存放在数组 num 中的原始数据为：

34	56	93	−8	0	120	88	−28	98	10

第一步：将 num[0]的值依次与 num[1]～num[9]（用 num[j]表示）相比较，如果 num[j]比 num[0]小，则交换 num[0]与 num[j]的值。第一轮交换的结果为：

−28	56	93	34	0	120	88	−8	98	10

显然，经过第一轮的比较，10 个数中的最小值−28 就被放在了第一个元素（num[0]）的位置。

第二步：将 num[1]的值依次与 num[2]～num[9]（用 num[j]表示）相比较，如果 num[j]比 num[1]小，则交换 num[1]与 num[j]的值。第二轮交换的结果为：

−28	−8	93	56	34	120	88	0	98	10

经过第二轮的比较，剩下 9 个数中的最小值−8 被放在了第二个元素（num[1]）的位置。

第三步：将 num[2]的值依次与 num[3]～num[9]（用 num[j]表示）相比较，如果 num[j]比 num[2]小，则交换 num[2]与 num[j]的值。第三轮交换的结果为：

−28	−8	0	93	56	120	88	34	98	10

经过第三轮的比较，剩下 8 个数中的最小值 0 被放在了第三个元素（num[2]）的位置。

……

可以看出，如果要排序的数据个数为 n，则应该比较 n−1 轮。

源程序如下：

```
main()
{
  int num[10],t;
  int i,j;
  clrscr();
  printf("输入 10 个整数：\n");
  for (i=0 ; i<10 ; i++)
     scanf("%d",&num[i]);
  for (i=0 ; i<9 ; i++)
     for (j=i+1 ; j<10 ; j++)
        if (num[j]<num[i])
           {
            t=num[i];
            num[i]=num[j];
            num[j]=t;
           }
  printf("排序后的结果为：\n");
  for (i=0;i<10;i++)
     printf("%d   ",num[i]);
}
```

运行结果如下：

```
输入 10 个整数：
34   56   93   −8   0   120   88   −28   98   10 ↙
排序后的结果为：
−28   −8   0   10   34   56   88   93   98   120
```

6.2　二维数组的定义和引用

6.2.1　二维数组的定义

1. 二维数组的定义

定义二维数组的一般形式为：

```
类型说明符  数组名[常量表达式 1][常量表达式 2] ;
```

例如：

```
float a[5][10] ;
```

该语句定义了一个名为 a 的 5×10 的二维数组，数组的类型为 float 型，数组中共有 50 个元素。

说明：

（1）数组名后的常量表达式的个数称为数组的维数。每个常量表达式必须用方括号括起来，如下面是非法的定义方法：

```
float a[5,10] ;
```

（2）二维数组中元素的个数为：常量表达式 1×常量表达式 2。

（3）如果常量表达 1 的值为 n，常量表达式 2 的值为 m，则二维数组中第一个元素的下标为[0][0]，最后一个元素的下标为[n−1][m−1]。

（4）一维数组通常用来表示一行或一列数据，而二维数组则通常用来表示呈二维表排列的一组相关数据。例如下面的一个存放学生成绩的数据表：

学生姓名	语文成绩	数学成绩	C 语言成绩
周宇航	81	76	90
冯天乐	94	90	85
张一凡	78	65	58
牟思娴	79	83	70

如果想用一个数组存放各个学生各门学科的成绩，则可定义如下二维数组：

```
int score[4][3] ;
```

该数组中的各个元素分别为：

score[0][0]　score[0][1]　score[0][2]
score[1][0]　score[1][1]　score[1][2]
score[2][0]　score[2][1]　score[2][2]
score[3][0]　score[3][1]　score[3][2]

这里，可以用数组元素 score[0][0]、score[0][1]、score[0][2]分别存放第一个学生的三门学科成绩，数组元素 score[1][0]、score[1][1]、score[1][2]分别存放第二个学生的三门学科成绩，等等。

2. 二维数组的机内表示

二维数组中的各个元素在机内是按行的顺序存放的，即先存放第一行的元素，再存放第二行的元素，以此类推。

例如：

```
int score[4][3] ;
```

二维数组 score 中的各个元素在机内的存储顺序为：

存储区
score[0][0]
score[0][1]
score[0][2]
score[1][0]
score[1][1]
score[1][2]
⋮
score[3][0]
score[3][1]
score[3][2]

可以看出，无论是一维数组，还是二维数组或多维数组，在内存中都是线性存储的。事实上，我们可以把二维数组看作是一种特殊的一维数组，这个一维数组中的每一个元素又是一个一维数组。

由于 C 语言对二维数组或多维数组是按行的顺序存放的，所以，第一维的下标变化最慢，而最后一维（即最右边）的下标变化最快。理解这一点，将有助你更好地使用数组进行编程。

6.2.2　二维数组的引用

二维数组中数组元素的表示形式为：

```
数组名[下标 1][下标 2]
```

说明：

（1）与一维数组相同，二维数组元素的下标也可以是任何整型常量、整型变量或返回整型量的表达式。

（2）如果二维数组第一维的长度为 n，第二维的长度为 m，则引用该二维数组的元素时，第一个下标的范围为 0～n−1，第二个下标的范围为 0～m−1。例如：

```
int a[3][4];
```

则各个数组元素顺序为：

a[0][0]、a[0][1]、a[0][2]、a[0][3]、
a[1][0]、a[1][1]、a[1][2]、a[1][3]、
a[2][0]、a[2][1]、a[2][2]、a[2][3]。

【例 6.7】二维数组的引用。

```
main()
{
  int a[3][4];
  int i,j;
  clrscr();
  printf("输入二维数组中各元素的值：\n");
  for(i=0;i<3;i++)              /*  变量 i 表示第一维下标的变化  */
    for(j=0;j<4;j++)            /*  变量 j 表示第二维下标的变化  */
        scanf("%d",&a[i][j]);
  for(i=0;i<3;i++)              /*  顺序输出二维数组中各个元素的值  */
    {
      for(j=0;j<4;j++)
        printf("%5d",a[i][j]);
      printf("\n");
    }
}
```

运行结果如下：

```
输入二维数组中各元素的值：
67  86  58  65  89  98  72  75  80  89  50  72 ↙
  67  86  58  65
  89  98  72  75
  80  89  50  72
```

6.2.3　二维数组的初始化

可以使用下面的方法来初始化二维数组：

（1）分行给二维数组赋初值。例如：

```
int b[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

第一对花括号内的数值赋给数组 b 第一行的元素，第二对花括号内的数值赋给第二行的元素，……，以此类推。

（2）也可以把所有的数据都写在一对花括号内。例如：

```
int b[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

但这种初始化二维数组的方法不如第一种方法直观。

（3）可以只对二维数组的部分元素赋初值。例如：

```
int b[3][4]={{1},{2},{3}};
```

这时，b[0][0]的值为 1，b[1][0]的值为 2，b[2][0]的值为 3。

```
int b[3][4]={{1},{2,3}};
```

这时，b[0][0]的值为 1，b[1][0]的值为 2，b[1][1]的值为 3。

（4）如果对二维数组的全部元素赋初值，则定义二维数组时，第一维的长度可以省略，但第二维的长度不能省。例如：

```
int b[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

可以写成：

```
int b[ ][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

【例 6.8】将下表中的学生成绩赋给数组 score，并依次显示出每个学生各门学科的成绩。

学生姓名	语文成绩	数学成绩	C 语言成绩
周宇航	81	76	90
冯天乐	94	90	85
张一凡	78	65	58
牟思娴	79	83	70

源程序如下：

```
main()
{
 int score[4][3]={{81,76,90},{94,90,85},{78,65,58},{79,83,70}};
 int i,j;
 clrscr();
 printf("%12s%12s%12s\n","语文成绩","数学成绩","C 语言成绩");
 for(i=0;i<4;i++)
    {
     for(j=0;j<3;j++)
        printf("%12d",score[i][j]);
     printf("\n");
    }
}
```

运行结果如下：

```
    语文成绩      数学成绩      C 语言成绩
          81            76            90
          94            90            85
          78            65            58
          79            83            70
```

6.2.4　二维数组应用举例

【例 6.9】分别输入 4 个学生的语文成绩、数学成绩和 C 语言成绩，求每个学生的总成绩和平均成绩。

程序分析：

在前面的【例 6.3】中，分别使用了 3 个数组来存放每个学生的三门课程成绩。而在本例中，我们将使用一个二维数组来存放所有学生的全部成绩数据，这样，可以精简程序代码。

源程序如下：

```
#define N 4
main()
{
 int score[N][3],sum[N];   /* 数组 score 存放学生成绩，数组 sum 存放学生的总成绩 */
 int i,j;
 clrscr();
 for (i=0;i<N;i++)
   {
    sum[i]=0;
    printf("输入第%d 个学生的三门成绩：",i+1);
    for (j=0;j<3;j++)
       {
        scanf("%d",&score[i][j]);
        sum[i]=sum[i]+score[i][j];
       }
   }
 for (i=0;i<N;i++)
   {
    printf("第%d 个学生的总成绩和平均成绩分别为：%d，%.2f\n",i+1,sum[i],sum[i]/3.0);
   }
}
```

运行结果如下：

```
输入第 1 个学生的三门成绩：72   75   80 ↙
输入第 2 个学生的三门成绩：81   76   88 ↙
输入第 3 个学生的三门成绩：94   86   85 ↙
输入第 4 个学生的三门成绩：60   73   78 ↙
第 1 个学生的总成绩和平均成绩分别为：227，75.67
第 2 个学生的总成绩和平均成绩分别为：245，81.67
第 3 个学生的总成绩和平均成绩分别为：265，88.33
第 4 个学生的总成绩和平均成绩分别为：211，70.33
```

6.3　字符数组

存放数值型数据的数组为数值型数组，如整型数组、单精度型数组等等。而字符型数组则是指专门用来存放字符型数据的数组，其中的每个元素存放一个字符。字符数组既具有普通数组的一般性质，又具有某些特殊性质。

6.3.1 字符数组的定义和初始化

1. 字符数组的定义

定义字符数组与前面介绍的定义数值型数组的方法类似，例如：

```
char ch[10];
```

上面的语句定义了一个名为 ch 的字符数组，数组的长度为 10。注意，字符数组中的每一个元素只能存放一个字符，例如：

```
ch[0] = 'h';
ch[1] = 'a';
ch[2] = 'p';
```

2. 字符数组的初始化

（1）用单个的字符常量对字符数组初始化。例如：

```
char ch[ ] = {'h' , 'a' , 'p' , 'p' , 'y'};
```

由于花括号中有 5 个字符常量，所以系统将确定字符数组 ch 的长度为 5。初始化后，ch 数组中 ch[0]元素的值为'h'，ch[1]的值为'a'，……，ch[4]的值为'y'。

（2）用字符串常量对字符数组初始化。例如：

```
char ch[ ] = "happy" ;
```

虽然字符串 "happy" 中只包含了 5 个字符，但系统却将确定字符数组 ch 的长度为 6。这是因为在编译过程中，系统会自动在每一个字符串的末尾都加上一个空字符'\0'，来作为字符串的结束标志。经过上面的初始化之后，ch 数组的前 5 个元素的值分别为'h'、'a' 、'p'、'p'、'y'，ch[5]元素的值为'\0'。

6.3.2 字符数组的引用

与数值型数组相同，字符数组的引用也可以通过对数组元素的引用来实现。

【例 6.10】字符数组的引用。

```
main()
{
 char ch[ ] = {'I' , ' ' , 'a' , 'm' , ' ' , 'a' , ' ' , 's' , 't' , 'u' , 'd' , 'e' , 'n' , 't'};
 int i;
 clrscr();
 for(i = 0 ; i<14 ; i++)
    printf("%c",ch[i]);
}
```

运行结果如下：

```
I am a student
```

【例 6.11】输出一个上三角形。

```
main()
{
 char graph[4][7]={{' ',' ',' ',' ','*'},{' ',' ',' ','*','*','*'},{' ',' ','*','*','*','*','*'},{'*','*','*','*','*','*','*'}};
 int i,j;
 for (i=0;i<4;i++)
    {
```

```
for (j=0;j<7;j++)
    printf("%c",graph[i][j]);
printf("\n");
        }
}
```

运行结果如下：

```
    *
   ***
  *****
*******
```

6.3.3　字符串与字符数组

C 语言中，字符数组一个最重要的作用就是用来处理字符串。C 语言中有字符串常量，却没有字符串变量，字符串的输入、存储、处理和输出等操作，都必须通过字符数组来实现。

【例 6.12】输出一个字符串。

```
main()
{
 char ch[]="I am a student";
 clrscr();
 printf("%s",ch);
}
```

运行结果如下：

```
I am a student
```

注意下面的语句：

```
    printf("%s",ch);
```

"%s"表示以字符串的形式输出数据。这里，引用字符数组 ch 时，只给出了数组名，而没有使用下标，这是因为 C 语言中把数组名作为该数组的首地址，即数组中第一个元素的存储地址，当以字符串的形式输出字符数组 ch 中的内容时，系统会根据 ch 数组的首地址，自动从 ch[0]元素开始，顺序输出各个元素的值（字符形式），直到遇到字符串的结束标志'\0'为止。



注意

这种直接使用数组名，把字符数组当作一个整体来处理的做法，只适用于字符型数组，而不适用于数值型数组。

6.3.4　字符串的输入、输出和处理

1. 字符串的输入

用于字符串输入的常用函数有两个：scanf()和 gets()。

若有：

```
    char ch[5];
```

则可以用下面的方法将一个字符串输入到字符数组 ch 中：

（1）for(i = 0 ; i<5 ; i++)

```
        scanf("%c",&ch[i]);
```

这里，通过运用循环语句，依次为数组的每一个元素输入值。"%c"表示以字符的形式输入

数据。

（2）scanf("%s",ch);

"%s"表示以字符串的形式输入数据。注意，不能在数组名 ch 的前面加上取地址符&，因为数组名 ch 已经代表了数组的首地址。

以这种方法输入字符串时，除了输入的字符串本身的内容被存入到数组 ch 中，字符串末尾的结束标志'\0'也会被存入到数组中。

需要注意的是，用 scanf()函数以"%s"的形式输入字符串时，存入到字符数组中的内容开始于输入字符中的第一个非空白字符，而终止于下一个空白字符（包括：'\n'、'\t'、' '）。

例如：

```
char ch[6]
scanf("%s" , ch) ;
```

若输入：

```
How  are  you ↙
```

则数组 ch 中的实际内容如下：

ch[0]	ch[1]	ch[2]	ch[3]	ch[4]	ch[5]
H	o	w	'\0'		

（3）gets(ch);

gets()函数的作用是输入一个字符串，其调用的一般形式为：

```
gets(字符数组名);
```

与 scanf()函数使用"%s"输入字符串不同的是，gets 函数可以将输入的换行符之前的所有字符（包括空格）都存入到字符数组中，最后加上字符串结束标志'\0'。在程序中使用 gets 函数时，需要包含头文件“stdio.h”。

2. 字符串的输出

用于字符串输出的常用函数有两个：printf()和 puts()。

若有：

```
ar ch[ ] = "How are you" ;
```

则可以用下面的方法输出字符数组 ch 中的内容：

（1）for(i = 0 ; i<11 ; i++)

```
    printf("%c",ch[i]);
```

这种方法是分别引用字符数组中的每一个元素，一个一个地输出数组元素中的字符。

（2）printf("%s",ch);

这种方法是以字符串的形式，一次输出整个字符数组中的所有字符。

（3）puts(ch);

puts()函数的作用是输出一个字符串，其调用的一般形式为：

```
puts(字符数组名或字符串常量);
```

与 printf()函数不同的是，puts()函数输出字符串时，会自动在字符串的末尾输出一个换行符'\n'。在程序中使用 puts 函数时，需要包含头文件“stdio.h”。

【例 6.13】输入和输出字符串。

```
# include "stdio.h"
main()
```

```
{
 char str1[]="Hello",str2[20];
 clrscr();
 printf("请输入你的姓名：\n");
 gets(str2);
 printf("%s   %s !",str1,str2);
}
```

运行结果如下：

```
请输入你的姓名：
Xiang   ua ↙
Hello   Xiang   Hua !
```

【例 6.14】原文变密码。

程序说明：

此例中，原文变密码的规则是：Z 变 X，z 变 x，即变成该字母前面的第二个字母，而 A 变 Y，a 变 y，B 变 Z，b 变 z。原文中不是字母的字符不变。

源程序如下：

```
# include "stdio.h"
main()
{
 char str1[80],str2[80];   /* str1 存放原文字符串，str2 存放密码字符串 */
 char ch;
 int i;
 puts("请输入原文：");
 gets(str1);
 i=0;
 while (str1[i]!='\0')     /* 开始扫描原文字符串，直到遇到字符串结束标志'\0' 为止 */
    {
     ch=str1[i];           /* 将扫描到的当前字符存放到字符变量 ch 中 */
     if (ch>='c' && ch<='z' || ch>='C' && ch<='Z')     /* 判断 ch 是否为 c～z 或 C～Z 中的字母之一 */
         ch=ch−2;
     else if (ch>='a' && ch<='b' || ch>='A' && ch<='B')     /* 对前两个字母作处理 */
         ch=ch+24;
     str2[i]=ch;       /* 将处理后的字符存入密码字符串中 */
     i++;              /* 扫描下一个字符 */
    }
 str2[i]='\0';         /* 扫描结束后，在密码字符串的末尾加上字符串结束标志'\0' */
 puts("密码为：");
 printf("%s",str2);
}
```

运行结果如下：

```
请输入原文：
I am a student ↙
密码为：
G yk y qrsbclr
```

3. 常用字符串处理函数

字符串的处理是程序设计中常遇到的问题，C 语言提供了许多专门用于处理字符串的函数，下面重点介绍最常用的 strlen 函数、strcat 函数、strcmp 函数和 strcpy 函数。使用这些函数时，需要包含头文件“string.h”。

（1）strlen()函数

strlen()函数的作用是测试字符串的长度，其调用的一般形式为：

```
strlen(字符数组名或字符串常量)
```

该函数的返回值即为字符串的长度。注意，字符串的长度并不包括字符串的结束标志'\0'。

【例 6.15】strlen()函数应用示例。

下面程序的功能是，输入一个字符串，然后按相反的顺序输出各个字符。

```
# include "stdio.h"
# include "string.h"
main()
{
 char str[80];
 clrscr();
 printf("输入一个字符串：\n");
 gets(str);
 printf("该字符串的长度为：%d",strlen(str));
}
```

运行结果如下：

```
输入一个字符串：
How are you! ↙
该字符串的长度为：12
```

（2）strcat()函数

strcat()函数的作用是连接两个字符串，其调用的一般形式为：

```
strcat(字符数组 1，字符数组 2);
```

strcat()函数把字符数组 2 连接到字符数组 1 的后面，连接的结果仍放在字符数组 1 中。需要注意的是，定义字符数组 1 时，其长度应该足够大，否则就没有多余的空间来存放连接后产生的新字符串。

【例 6.16】strcat()函数应用示例。

```
# include "stdio.h"
# include "string.h"
main()
{
 char str1[30],str2[20];
 clrscr();
 printf("请输入第 1 个字符串：");
 gets(str1);
 printf("请输入第 2 个字符串：");
 gets(str2);
 strcat(str1,str2);
 printf("连接后的字符串为：%s", str1);
```

```
}
```

运行结果如下：

```
请输入第 1 个字符串：I  am  a ↙
请输入第 2 个字符串：  student ↙
连接后的字符串为：I  am  a  student
```

（3）strcmp()函数

strcmp()函数的作用是比较两个字符串是否相等，其调用的一般形式为：

```
    strcmp(字符串 1，字符串 2)
```

如果字符串 1 = 字符串 2，则函数返回 0。

如果字符串 1 > 字符串 2，则函数返回正数。

如果字符串 1 < 字符串 2，则函数返回负数。

【例 6.17】strcmp()函数应用示例。

```
# include "stdio.h"
# include "string.h"
main()
{
 char str1[80],str2[80];
 clrscr();
 printf("输入第一个字符串：");
 gets(str1);
 printf("输入第二个字符串：");
 gets(str2);
 if (strcmp(str1,str2) = = 0 )
    printf("两个字符串内容相同");
 else
    printf("两个字符串内容不同");
}
```

运行结果如下：

```
输入第一个字符串：Hello ↙
输入第二个字符串：Welcome ↙
两个字符串内容不同
```

（4）strcpy()函数

strcpy()函数的作用是复制字符串，其调用的一般形式为：

```
    strcpy(字符数组 1，字符串 2)
```

strcpy()函数把字符串 2 的内容复制到字符数组 1 中。这里，字符串 2 可以是字符数组名，也可以是字符串常量，而字符数组 1 则只能是字符数组名。



注意

不能使用赋值语句将一个字符串常量或字符数组直接赋值给一个字符数组，例如下面的语句是错误的：

```
    str1 = "hello" ;
    str1 = str2 ;
```

【例 6.18】strcpy()函数应用示例。

下面程序的功能是交换两个字符数组 str1 和 str2 的内容。

```
# include "stdio.h"
```

```
# include "string.h"
main()
{
 char str1[20],str2[20],temp[20];
 clrscr();
 printf("输入字符串 str1：");
 gets(str1);
 printf("输入字符串 str2：");
 gets(str2);
 strcpy(temp,str1);
 strcpy(str1,str2);
 strcpy(str2,temp);
 printf("交换后 str1 的内容为：%s\n",str1);
 printf("str2 的内容为：%s",str2);
}
```

运行结果如下：

```
输入字符串 str1：abc ↙
输入字符串 str2：1234 ↙
交换后 str1 的内容为：1234
str2 的内容为：abc
```



习题六

1. 填空题

（1）数组是一组具有相同_____的数据的集合。

（2）在C语言中，没有字符串变量，字符串的存储是通过_____来实现的。

（3）strlen函数的功能是_____，strcmp函数的功能是_____。

2. 选择题

（1）定义一个有 100 个元素的int型数组，下面正确的语句是_____。

A. int a(99);　　B. int a[99];

C. int a(100);　　D. int a[100];

（2）下面正确的对数组 b 进行初始化的语句是_____。

A. int b[10]=1;　　B. int b[10]=(1,2,3);

C. int b[]={1,2,3};　　D. int b[10]={}

（3）在 C 程序中，引用一个数组元素时，其下标的数据类型允许是_____。

A. 任何类型的表达式　　B. 整型常量

C. 整型表达式　　D. 整型常量或整型表达式

（4）下面语句中正确的是_____。

A. char name []={'T','o','m'};　　B. char name="Tom";

C. char name[3]="Tom";　　D. char name[]='T','o','m','\0' ;

3. 分析下列程序，写出运行结果

（1）main()

```
{
  int a[10],i;
  for(i=0;i<10;i++)
     {
       a[i]=i+1;
       printf("a[%d]=%d\n",i,a[i]);
     }
}
```

（2）main()

```
{
  int a[5]={10,20,30,40,50};
  int b[5]={1,2,3};
  int c[]={0,1,2,3};
  int i;
  printf("数组 a：");
  for(i=0;i<5;i++)
     printf("%5d",a[i]);
  printf("\n");
  printf("数组 b：");
  for(i=0;i<5;i++)
     printf("%5d",b[i]);
  printf("\n");
  printf("数组 c：");
  for(i=0;i<4;i++)
     printf("%5d",c[i]);
}
```

4. 编程题

（1）输入一组数，求其中的最大值和最小值，以及这组数的和及平均值。

（2）求 100 以内的所有素数，每行输出 8 个。

（3）输入二维数组 a[4][6]，输出其中的最大值及其对应的行列位置。

（4）将一个字符串插入到另一个字符串的指定位置。例如，将字符串"abc"插入到字符串"123456"中的第三个位置，则插入后的结果应为"12abc3456"。



上机实习指导

一、学习目标

本章重点介绍了 C 语言中的数组及其有关操作。数组是几乎每一种高级语言都提供了的数据类型，它是我们学习 C 语言程序设计必须重点掌握的基础知识之一。

通过本章的学习，应达到以下要求：

1. 熟练掌握数值型数组（包括一维数组和二维数组）、字符型数组的定义和引用方法。
2. 熟练掌握数值型数组和字符型数组的初始化方法。
3. 掌握常用的字符串处理函数。

4. 理解选择排序算法。

5. 在实际编程中能灵活运用数组解决相关问题。

二、定义和引用数组时易出现的错误

1. 定义数组时没有指定数组的长度。

只有在两种情况下，定义数组时才可以不指定数组的长度，一是定义数组的同时初始化数组，二是定义作为形式参数的数组。除此之外，定义数组时必须指定数组的长度，否则，程序在编译时系统将提示如下出错信息：

```
Size of structure or array not known
```

2. 定义数组时用变量表示数组的长度。

定义数组时数组的长度只能用常量表达式表示，否则，程序编译时 Message 窗口中将提示如下信息：

```
Constant expression required
```

3. 引用数组元素时下标超出范围。

对一个长度为 n 的数组，引用数组元素时下标的范围是 0 至 n−1。当下标超出范围时，程序在编译和连接时都不会产生任何出错信息，但程序运行时就可能产生莫名其妙的中断或死机。因此，引用数组元素时下标超出范围的错误既危险又难以查找，在上机调试程序时一定要多加注意。

4. 直接把字符串赋值给一个字符数组。

不能用赋值的形式把字符串存入一个字符数组中（定义字符数组的同时初始化数组除外），正确的做法是调用字符串复制函数 strcpy()。

下面程序段的写法是正确的：

```
char s[6]="Hello";
……
puts(s);
```

或

```
char s[6];
……
strcpy(s,"Hello");
puts(s);
```

下面的程序段是错误的：

```
char s[6];
……
s="Hello";
puts(s);
```

5. 直接用关系运算符“==”比较两个字符串是否相等。

要比较两个字符串的内容是否相同，就应该使用 strcmp()函数，而不能用关系运算符 “==”直接比较。

看下面的程序：

```
main()
{
 char s1[6]="Hello",s2[6]="Hello";
 clrscr();
 if (s1= =s2) printf("两个字符数组的内容相同");
```

```
    else printf("两个字符数组的内容不同");
}
```

运行结果：

```
    两个字符数组的内容不同
```

虽然字符数组 s1 和 s2 的内容完全一样，但程序的运行结果仍然显示出其内容不同的信息，事实上，无论 s1 和 s2 的内容是否相同，程序运行时都会产生这样的运行结果。

为什么会这样呢？这是因为关系表达式 s1= =s2 并非判断字符数组 s1 和 s2 的内容是否相同，而是判断数组名 s1 和 s2 的值（即数组的首地址）是否相等！由于这是两个不同的数组，因此关系表达式 s1= =s2 始终为假。

所以，要比较字符数组 s1 和 s2 的内容是否相同，就应该把上面程序中的 if 语句改写如下：

```
if (strcmp(s1,s2) = =0 ) printf("两个字符数组的内容相同");
else printf("两个字符数组的内容不同");
```

或

```
if (! strcmp(s1,s2) ) printf("两个字符数组的内容相同");
else printf("两个字符数组的内容不同");
```



上机实习一　数值型数组

一、目的要求

1. 熟练掌握数值型数组的定义和引用方法。
2. 熟练掌握数值型数组的初始化方法。
3. 在实际编程中能灵活运用数组处理一组具有共性的数据。
4. 在调试程序的过程中，逐步熟悉一些与数组有关的出错信息，提高程序调试技巧。

二、上机内容

1. 上机调试下面的程序，修改其中存在的错误。

（1）main()

```
{
 int b=5,i;
 int a[b]={1,2,3,4,5};
 for (i=0;i<b;i++)
    printf("%d    ",a[i]);
}
```

（2）main()

```
{
 int a[ ];
 int i,sum;
 for (i=0;i<=10;i++)
    {
      scanf("%d" , &a[i]);
      sum+=a[i];
    }
 printf("sum=%d", sum);
}
```

2. 运行下列程序，分析并观察运行结果。

```
main()
{
 int a[10]={89,67,100,64,76,90,94,52,82,90};
 int num,i;
 clrscr();
 printf("输入要查找的数：");
 scanf("%d",&num);
 for(i=0 ; i<10 ; i++)
    if (a[i] = = num) break;
 if (i<10) printf("%d 在这组数中的第%d 个位置。",num , i+1);
 else printf("%d 不在这组数中。" , num);
}
```

说明：

运行这个程序时分别输入下面三个数据，注意观察各自的输出结果。

输入数据一：

```
    89 ↙
```

输入数据二：

```
    120 ↙
```

输入数据三：

```
    94 ↙
```

3. 完善程序。

根据程序的功能，在程序中的横线处填写正确的语句或表达式，使程序完整。上机调试程序，使程序的运行结果与给出的结果一致。

（1）输入 20 个整数，统计其中非负数个数，并求非负数之和。

```
main()
{
 int i,num[20];
 int count,sum;
 count=0; sum=0;
 printf("输入 20 个整数：\n");
 for(i=0;i<20;i++)
    {
     scanf("%d", _____);
     if   (num[i] _____)
            {count++; sum=_____;}
    }
 printf("非负数的个数为：%d\n",count);
 printf("非负数之和为：%d",sum);
}
```

（2）输入一组学生的语文成绩和数学成绩，求每个学生的平均成绩。要求按后面运行结果所示的格式输出数据。

```
main()
{
 int score[40][3],i,j,num;
 float av[40];
```

```
  clrscr();
  printf("输入学生人数（不超过 40 人）：");
  scanf("%d",&num);
  for(i=0;i<num; _____)
     {printf("输入第%d 个学生的语文成绩和数学成绩：",i+1);
      for(j=0; _____;j++)
         scanf("%d",&score[i][j]);
     }
  for(i=0;i<num;i++)
     {score[i][2]=0;      /*   score[i][2]存放总成绩   */
      for(j=0;j<2;j++)
         _____;      /*   求总成绩   */
      av[i]=                ;      /*   求平均成绩   */
     }
  printf("%8s%10s%10s%10s%10s\n","编号","语文成绩","数学成绩","总成绩","平均成绩");
  for(i=0;i<num;i++)
     {printf("%8d", _____);          /*   输出编号   */
      for(j=0;j<3;j++)
         printf("%10d", _____);   /*   输出语文、数学和总成绩   */
      printf("%10.1f\n", _____);   /*   输出平均成绩   */
     }
}
```

运行结果：

```
输入学生人数（不超过 40 人）：4 ↙
输入第 1 个学生的语文成绩和数学成绩：86   83 ↙
输入第 2 个学生的语文成绩和数学成绩：75   81 ↙
输入第 3 个学生的语文成绩和数学成绩：90   87 ↙
输入第 4 个学生的语文成绩和数学成绩：65   74 ↙
    编号    语文成绩    数学成绩    总成绩    平均成绩
       1          86          83       169        84.5
       2          75          81       156        78.0
       3          90          87       177        88.5
       4          65          74       139        69.5
```



上机实习二　字符型数组

一、目的要求

1. 熟练掌握字符数组的定义和引用方法。
2. 熟练掌握字符数组的初始化方法。
3. 掌握常用的字符串处理函数。

二、上机内容

1. 上机调试下面的程序，修改其中存在的错误。

（1）main()

```
    {
```

```
    char str[10];
    str="Hello";
    puts(str);
  }
```

（2）main()

```
  {
    char str1[20],str2[20];
    gets(str1);
    gets(str2);
    if   (str1= =str2) printf("两个字符串相同");
    else printf("两个字符串不同");
  }
```

2. 完善程序。

根据程序的功能，在程序中的横线处填写正确的语句或表达式，使程序完整。上机调试程序，使程序的运行结果与给出的结果一致。

（1）下面是一个简单的输入密码程序，可判断输入的字符串是否与预先设置的密码相同。

```
# include "stdio.h"
# include "string.h"
main()
{
 char password[ ] = "hello";     /* 预先设置密码 */
 char str[20];
 clrscr();
 printf("请输入密码：\n");
 gets(str);
 if   (_____)
    printf("密码正确！ ");
 else
    {
     printf("_____");
     exit(0);     /* 终止程序的运行 */
    }
 printf("请继续……");
}
```

运行结果之一：

```
请输入密码：
hello ↙
密码正确！请继续……
```

运行结果之二：

```
请输入密码：
abcde ↙
密码不正确！
```

（2）输入一组学生的姓名和成绩，根据成绩排出名次，要求按后面运行结果所示的格式输出数据。

```
# include "stdio.h"
# include "string.h"
```

```
main()
{
 char name[40][10] , str[10];
 int score[40] , num , i , j , t;
 clrscr();
 printf("输入学生人数：");
 scanf("%d" , &num);
 for(i=0 ; i<num ; i++)
    {
     printf("输入第%d 位学生的姓名和成绩：" , i+1);
     scanf("%s %d" , name[i] , &score[i]);
    }
 for(i=0 ;                    ; i++)          /*   排序   */
    for(j=_____;j<num;j++)
       if (score[j]>score[i])
             {t=score[i];
              _____;
              _____;
              strcpy(str , name[i]);
              _____;
              _____;
             }
 printf("排了名次的成绩单如下：\n");
 printf("%8s%12s%8s\n" , _____);
 for(i=0;i<num;i++)
    printf("%8d%12s%8d\n" , i+1 , _____, score[i]);
}
```

运行结果：

```
    输入学生人数：4 ↙
    输入第 1 位学生的姓名和成绩：赵勇   76 ↙
    输入第 2 位学生的姓名和成绩：刘华   73 ↙
    输入第 3 位学生的姓名和成绩：张小宇   88 ↙
    输入第 4 位学生的姓名和成绩：冯向   100 ↙
    排了名次的成绩单如下：
      名次          姓名          成绩
        1          冯向           100
        2        张小宇            88
        3          赵勇            76
        4          刘华            73
```

第 7 章　函　　数

函数是实现了一定功能的、具有一定格式的程序段，是 C 语言程序的基本组成单位。利用函数可以实现程序的模块化，使得复杂问题得以轻松解决。本章主要介绍函数的定义和调用方法，以及与函数有关的一些基本概念。

【本章要点】

1. 函数的定义和调用；
2. 函数参数和函数的返回值；
3. 变量的作用域。

【学习目标】

1. 了解 C 语言函数的分类；
2. 掌握函数的定义和调用方法；
3. 了解局部变量和全局变量的概念及其作用范围；
4. 在实际编程中能够合理地使用不同作用域的变量。

【课时建议】

讲授 6 课时，上机 4 课时。

7.1　函数概述

7.1.1　为什么要使用函数

程序设计时，如果遇到一个复杂的问题，那么最好的方法就是将原始问题分解成若干个易于求解的小问题，每一个小问题都用一个相对独立的程序模块来处理，最后，再把所有的模块像搭积木一样拼合在一起，形成一个完整的程序。这种程序设计中分而治之的策略，被称为模块化程序设计方法，这是结构化程序设计中的一条重要原则。

几乎所有的高级程序设计语言都提供了自己的实现程序模块化的方法（如子程序、过程和函数等）。C 语言中，由于函数是程序的基本组成单位，所以，可以很方便地利用函数实现程序的模块化，这也是 C 语言的重要特色之一。

例如，要设计一个算术练习程序，要求这个程序能随机给出加、减、乘、除 4 种算术练习题，并能判断答题者的答案是否正确。根据程序的功能，可以把整个程序分成五大模块，其中，一个模块实现加、减、乘、除 4 种运算的菜单选择功能，另外四个模块分别实现 4 种运算的出题和对答案正误的判断。这里，菜单模块可用主函数来实现，通过主函数对另外 4 个函数的调用来把它们拼合起来。图 7-1 表示了这个程序的模块结构，在本章末尾的上机实习三中，将详细介绍这个算术练习程序的设计思路，并给出部分源程序清单。

显然，利用函数不仅可以实现程序的模块化，使用程序设计变得简单和直观，同时，也提

高了程序的易读性和易维护性。而且，我们还可以把程序中需要多次执行的计算或操作编写成通用的函数，以备需要时调用。同一函数不论在程序中被调用多少次，在源程序中只需书写一次，编译一次，这样，就避免了大量的重复程序段，缩短了源程序的长度，也节省了占用的内存空间，减少了编译时间，真是一举数得！

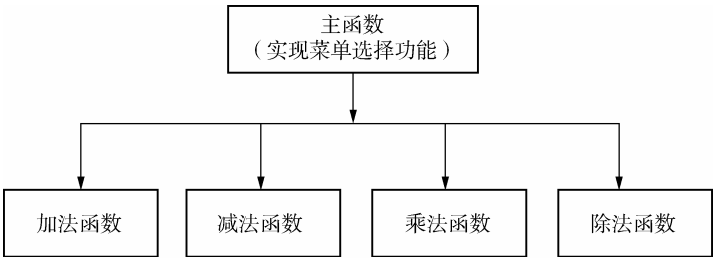


图 7-1　算术练习程序的模块结构图

7.1.2　库函数和用户自定义函数

C 语言函数分成系统提供的库函数和用户自定义函数两大类。库函数是系统已经设计好的函数，Turbo C 提供了 300 多个库函数（详见附录 B），编程时可直接调用库函数来完成各种各样的任务。例如，前面介绍的格式输入/输出函数 scanf 和 printf 函数，以及 strcat、strcpy 和 strlen 等字符串处理函数，都是库函数。

除了库函数之外，用户还可以根据自己的需要定义用以解决具体问题的函数，然后通过函数调用来实现所需的功能。

本章主要介绍用户自定义函数。

7.2　函数的定义

7.2.1　函数定义的一般形式

函数的定义主要是确定函数的名称、函数的类型以及该函数完成什么功能。函数定义的一般形式如下：

```
类型标识符　函数名（形式参数表）
形式参数说明
{
 函数体
}
```

下面，先看一个简单的例子。

【例 7.1】定义一个 sum 函数，求两个整数之和。

```
main()
{
 int num1 , num2 ;
 int s ;
 clrscr() ;
 printf("输入两个整数：") ;
 scanf("%d,%d",&num1,&num2) ;
 s = sum(num1,num2) ;   /* 调用 sum 函数，并把得到的函数返回值存入变量 s 中 */
```

```
 printf("两个数之和为：%d",s) ;
}
int sum(x,y)      /* 定义 sum 函数，函数类型为 int，形式参数为 x 和 y */
int x,y ;          /* 说明形式参数的类型 */
{
 int z;
 z=x+y;
 return (z) ;      /* 返回变量 z 的值 */
}
```

运行结果如下：

```
输入两个整数：27,10 ↙
两个数之和为：37
```

这个程序由两个函数组成，一个是主函数 main()，一个是自定义函数 sum()。sum()函数的功能是求 x 和 y 之和，并返回求得的和。main()函数调用自定义函数 sum()时，将实际参数 num1 和 num2 的值分别传递给形式参数 x 和 y。

7.2.2 有关函数定义的几点说明

1. 函数的类型

定义函数时，函数名前的类型标识符说明了函数的类型，这也是该函数的返回值的数据类型。类型标识符可以是 int、long、float、double、char 中的任何一种，当函数类型为 int 型时，类型标识符 int 可以省略。例如，【例 7.1】中的

```
int sum(x,y)
```

可以写成：

```
sum(x,y)
```

2. 函数名

函数名的命名遵循标识符规则，为了提高程序的易读性，定义函数时，最好给函数取一个见名知意的名字，也就是说，一个好的函数名能够反映该函数的功能。

3. 形式参数表

定义函数时，函数名后面圆括号中的变量名称为形式参数，简称“形参”。如果形式参数不止一个，那么每个形式参数之间则以逗号分隔。

需要注意的是，并不是每个函数在定义时都必须有形参，但无论有没有形参，函数名后面的圆括号都不能省略。

4. 形式参数的说明

如果函数有形式参数，则必须说明形式参数的类型。下面两种说明形式参数类型的写法都是正确的：

```
sum(x , y)
int x , y ;
{  }
```

和

```
sum(int x , int y)
```

```
{  }
```

注意比较这两种写法，在圆括号之外说明形参的类型时，如果几个形参都是属于同一种类型，那么就可以只用一个类型标识符同时说明这几个形参。而在圆括号之内说明形参的类型时，则不论各个形参是否属于同一类型，每个形参之前都必须有一个类型标识符。

5. 函数体

函数体是用一对花括号括起来的语句序列，函数的功能就是由这些语句共同完成。所有在函数体中使用到的形式参数之外的变量，都可以在函数体的开始部分进行变量的类型说明。如在【例 7.1】的 sum()函数中，定义了变量 z。

6. 空函数

定义函数时，函数类型、形式参数以及函数体均可以省略。所以，最简单的函数定义是：

```
函数名( )
{  }
```

这种函数称为空函数。显然，空函数不执行任何操作，但这并不意味着空函数是没有用处的。事实上，在编写程序的最初阶段，空函数非常有用，在设计一个较复杂的程序时，通常不可能一步到位地编写好每个功能模块，这时就可以利用空函数来表示没有编写好的模块，以确保程序结构的完整，使程序在最初的调试中能顺利地通过语法检测。以后，再根据需要在每个空函数内添加上具体的内容，逐步扩充程序的功能。

7. 自定义函数在程序中的位置

一个 C 程序由主函数和若干个自定义函数组成，各个函数在程序中的定义是相互独立的，不能在一个函数的函数体内部定义另一个函数。

自定义函数可以放在主函数之前，也可以放在主函数之后，但无论自定义函数放在程序中的什么位置，程序的执行总是从主函数开始的。为了提高程序的可读性，习惯上常常把主函数放在所有自定义函数之前。

7.3　函数参数及返回值

7.3.1　函数参数

1. 为什么要使用参数

对编程的初学者而言，往往不清楚一个函数什么情况下需要有形式参数。下面，我们通过两个具体的程序加以说明。

【例 7.2】编写一个函数，打印一串星号。

```
main()
{
  clrscr() ;
  pstar() ;
  printf("   欢迎使用本程序！\n") ;
  pstar() ;
}
pstar()      /* 定义 pstar 函数 */
```

```
{
 printf("********************\n") ;
}
```

运行结果如下：

```
********************
   欢迎使用本程序！
********************
```

这个程序中，main()函数两次调用 pstar()函数都打印出数目相同的一串星号，如果想打印不同数量的星号，则最容易想到的方法就是分别定义打印不同数量星号的函数，但显然，这种方法是不可取的。

【例 7.3】编写一个函数，打印一串星号，星号的数量由参数决定。

```
main()
{
 int i;
 clrscr() ;
 for (i=0;i<=4;i++)
      pstar(i*2+1) ;
}
pstar(num)
int num ;          /*  形式参数 num 表示要打印的星号的个数  */
{
 int i ;
 for (i=1 ; i<=num ; i++)
    printf("*") ;
 printf("\n") ;
}
```

运行结果如下：

```
*
***
*****
*******
*********
```

这个程序的功能是打印一个由星号组成的三角形图案。这个程序中，pstar()函数同样是执行打印一串星号的功能，但每次调用 pstar()函数都可以打印出不同数量的星号，星号数量的变化通过形式参数 num 来实现。在 main()函数中调用 pstar()函数时，只需将希望得到的星号的数量值，以实际参数的形式传递给 pstar()函数即可。

通过上面的两个程序可以看出，使用形式参数的根本目的，是为了提高一个函数的灵活性和通用性。

2. 形式参数和实际参数

形式参数是指定义函数时，跟在函数名后的小括号内的变量名。实际参数（简称“实参”）则是指调用函数时，跟在函数名后的小括号内的表达式。

实际参数与形式参数的关系如下：

（1）实际参数的数量应该与形式参数相同。如果一个函数没有形式参数，则调用该函数时就不应有实际参数，如，【例 7.2】中对 pstar 函数的定义和调用就是如此。

（2）实际参数的类型应该与形式参数一致。

（3）定义函数时的形式参数只能是变量名，而调用函数时的实际参数则可以是变量名，也可以是常量或表达式。如【例 7.3】，在主函数中调用 pstar()函数时，就使用了一个表达式来作为实际参数。

（4）当简单变量作函数参数时，参数的传递是“值传递”，这是一种单向传递，即数据只能由实参传给形参，而不能由形参传回给实参。看下面的例子：

【例 7.4】参数的传递。

```
main()
{
  int a=3;
  printf("a=%d\n",a);
  change(a);
  printf("a=%d",a);
}
change(int x)
{
  x=x+10;     /* 改变形参 x 的值 */
  printf("x=%d\n",x);
}
```

运行结果如下：

```
a=3
x=13
a=3
```

上例中，虽然在 change 函数中形参 x 的值发生了改变，但主函数中实参 a 的值却没有发生变化。【例 7.4】的参数传递情况如图 7-2 所示。

从图中可以看出，形参变量和实参变量分别占用不同的存储单元，所以，无论形参的值如何变化，都不会影响到实参的值。

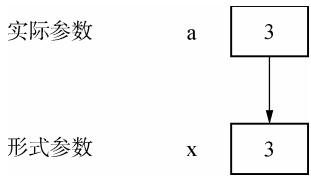


图 7-2　参数传递示意图

7.3.2　函数的返回值

函数的返回值用 return 语句实现，例如，在【例 7.1】的 sum 函数中，使用 return 语句返回了变量 z 的值。return 语句的一般形式如下：

```
return (表达式) ;
```

或

```
return 表达式 ;
```

或

```
return ;
```

例如：

```
return (1) ;        返回值为 1
return (a+b) ;      返回表达式 a+b 的值
return (x) ;        返回变量 x 的值
```

return 语句的作用有两个，一是终止包含它的那个函数的运行，使程序返回到调用该函数的语句处继续执行；二是用来送回一个数据，这个数据就是紧跟在 return 之后的表达式的值，也就是函数的返回值。在主调函数中，可以引用由 return 语句带回的函数的返回值。

说明：

1. 并不是每一个自定义函数都必须有 return 语句，如果一个函数不需要带回任何数据，那么这个函数可以没有 return 语句。

2. 一个没有 return 语句的函数，并不意味着没有返回值。实际上，任何一个类型不为 void 的函数都有一个返回值，包含 return 语句的函数将返回一个确定的值，而没有包含 return 语句的函数则返回一个不确定的值。

3. 可以引用不含 return 语句的函数所带回的不确定的返回值，这不会出现任何语法错误，但这种做法是毫无意义的，而且，还有可能使程序的执行产生难以预料的后果。因此，为了禁止引用不带 return 语句的函数的值，可在定义函数时指定函数的类型为 void 型，即空值类型。例如：

```
void pf ()
{
  ……
}
```

4. 函数中可以有多个 return 语句，但这并不意味着一个函数可以同时返回多个值。当执行到被调函数中的第一个 return 语句时，就会立即返回到主调函数。也就是说，只有一个 return 语句才有机会被执行。

7.4 函数的调用

7.4.1 函数的语句调用

函数的语句调用是把函数调用作为一个语句。其一般形式为：

```
函数名(实参表) ;
```

这种调用方式通常用于调用一个不带回返回值的函数，如，【例 7.2】和【例 7.3】中对函数 pstar()的调用方式就属于语句调用。

如果调用的函数无形式参数，则实参表可以没有，但函数名后面的小括号不能省去。下面是一个函数语句调用的例子：

【例 7.5】编写一个函数，打印两个数中的最小值。

```
main()
{
  int a,b ;
  clrscr() ;
  printf("a=") ;
  scanf("%d",&a) ;
  printf("b=") ;
  scanf("%d",&b) ;
  min(a,b) ;      /* 函数的语句调用 */
}
min(num1,num2)
int num1,num2 ;
{
  int m ;
  if (num1<num2) m=num1 ;
```

```
  else m=num2 ;
  printf("最小值为：%d",m) ;
}
```

运行结果如下：

```
a=13 ↙
b=86 ↙
最小值为：13
```

7.4.2　函数表达式调用

函数可以出现在表达式中，这种表达式称为函数表达式。其一般形式为：

```
    变量名=函数表达式
```

这种调用方式用于调用带有返回值的函数，函数的返回值将参加表达式的运算。如，【例 7.1】中对 sum()函数的调用，就是属于函数表达式调用。

下面的程序用另一种方式来实现【例 7.5】的功能。

【例 7.6】编写一个函数，求两个数中的最小值。

```
main()
{
  int a,b,n ;
  clrscr() ;
  printf("a=") ;
  scanf("%d",&a) ;
  printf("b=") ;
  scanf("%d",&b) ;
  n=min(a,b) ;      /* 函数的表达式调用 */
  printf("最小值为：%d",n) ;
}
int min(num1,num2)
int num1,num2 ;
{
  int m ;
  if (num1<num2) m=num1 ;
  else m=num2 ;
  return m;
}
```

运行结果如下：

```
a=13 ↙
b=86 ↙
```

最小值为：13

说明：

对于一个带有返回值，并且返回值的类型不为 int 型的函数，定义该函数时应该指明该函数的类型，即，函数名之前应该有类型标识符。同时，在调用该函数之前，还应该在主调函数中声明被调函数的类型。看下面的例子：

【例 7.7】调用函数求 n!。

```
main()
{
```

```
 int n ;
 long t ;
 long f() ;        /* 声明被调函数 f()的类型为 long 型 */
 clrscr() ;
 printf("输入一个整数：\n") ;
 scanf("%d" , &n) ;
 t=f(n) ;
 printf("%d!=%ld" , n , t) ;
}
long f(num)
int num ;
{
 long x ;
 int i ;
 x=1 ;
 for (i=1 ; i<=num ; i++)
    x*=i ;
 return x ;
}
```

运行结果如下：

```
输入一个整数：8 ↙
8! = 40320
```

注意主函数中的函数声明语句：

```
    long f() ;
```

该语句声明了 f()函数的返回值类型为 long 型。在主调函数中对被调函数作类型声明，意在告诉编译系统，本函数中将要用到的某函数是什么类型，以便让编译系统作出相应的处理。

函数声明的一般形式为：

```
    类型标识符 函数名() ;
```

程序中的

```
    long t ;
    long f() ;
```

也可以写成：

```
    long t , f() ;
```



注意

函数的类型声明是函数调用中一个非常重要的环节，初学者往往因忽略它而导致在上机调试程序时遇到麻烦。

在上面的这个程序中，如果将主函数中的 long f()删去，则程序在编译时系统将提示如下出错信息：

```
    Type mismatch in redeclaration of ' f '
```

但是，并不是对每一个被调函数都必须声明其类型。在下面几种情况下，可以不对被调函数作类型声明：

1. 当被调函数的返回值是 int 型或 char 型时，可以不用声明其类型。

2. 被调函数的定义出现在主调函数之前时，可以不声明其类型。试比较下面同一程序的两种写法：

（1）被调函数在主调函数之后

```
main()
{
 long f() ;
 ……
 t=f() ;
 ……
}
long f()
{……}
```

（2）被调函数在主调函数之前

```
long f()
{……}
main()
{
 ……
 t=f() ;
 ……
}
```

3. 在源程序的开头，在所有函数定义之前已说明了函数的类型，那么，在每个主调函数中都不必再对被调函数作类型声明。例如：

```
long f() ;
main()
{
 ……
 t=f() ;
 ……
}
lont f()
{……}
```

7.4.3　函数的递归调用

C 语言中，允许函数直接或间接地调用自己，这种调用方式称为函数的递归调用。其一般形式为：

```
a(x)
{
 ……
 a(y) ;      /*  直接的递归调用  */
 ……
}
```

或

```
a(x)
{
 ……
 b() ;
 ……
}
```

```
b()
{
 ……
 a(y) ;        /* 间接的递归调用 */
 ……
}
```

下面看一个递归调用的例子：

【例 7.8】用递归调用的方法，求 x^n（x 和 n 均为正整数）。

```
main()
{
 int a , b ;
 long power() , t ;
 clrscr() ;
 printf("输入两个整数：") ;
 scanf("%d , %d" , &a , &b) ;
 t=power(a , b) ;
 printf("%d ^ %d = %ld" , a , b , t) ;
}
long power(x , n)
int x , n ;
{
 long y;
 if (n>0)
    y=x*power(x , n-1) ;      /* 直接的递归调用 */
 else y=1 ;
 return y ;
}
```

运行结果如下：

```
输入两个整数：3 , 4 ↙
3 ^ 4 = 81
```

power()函数在执行的过程中，通过 power(x , n–1)直接调用了它自己。程序中递归调用的条件是 n>0，当这个条件不再满足时（即 n=0 时），即终止了递归调用。

这个程序的执行过程如图 7-3 所示。

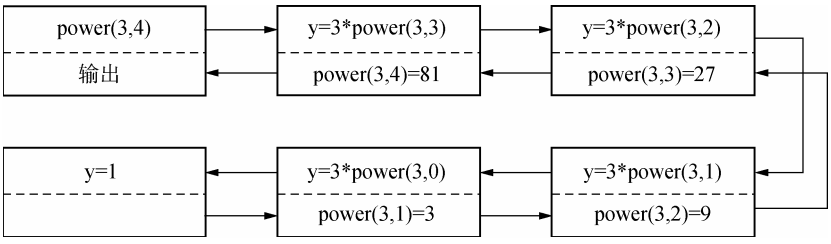


图 7-3 【例 7.8】程序的执行过程

递归调用的优点是使程序简洁、紧凑，但由于每次调用一个函数时都需要存储空间来保存调用“现场”，以便后面返回，并且递归调用往往涉及到同一个函数的反复调用，所以它要占用很大的存储空间，特别是在递归调用次数较多的情况下，将导致程序运行速度较慢。

7.4.4　函数的嵌套调用

函数的嵌套调用是指在调用一个函数的过程中，又去调用另一个函数。其一般形式为：

```
main()
{
  ……
  a();
  ……
}
a()
{
  ……
  b();
  ……
}
b()
{
  ……
}
```

这里，主函数调用了 a()函数，a()函数在执行的过程中又调用了 b()函数。每一个被调函数执行完之后，都将返回到调用该函数的地方继续往下执行。

【例 7.9】函数的嵌套调用。

```
main()
{
  int a,b;
  void head();
  clrscr() ;
  head() ;
  printf("a=") ; scanf("%d",&a) ;
  printf("b=") ; scanf("%d",&b) ;
  printf("a+b=%d", a+b);
}
void head()
{
  pstar() ;
  printf("     本程序的功能是求两个整数之和\n") ;
  pstar() ;
}
pstar()
{
  printf("**********************************\n") ;
}
```

运行结果如下：

```
**********************************
     本程序的功能是求两个整数之和
**********************************
```

```
a=13 ↙
b=86 ↙
a+b=99
```

程序中，main 函数调用了 head()函数，head()函数又调用了 pstar()函数。pstar()函数执行完之后，返回到调用它的 head()函数中，继续执行调用处后面的语句。同样，head()函数执行完之后，返回到调用它的 main 函数中。

在本章末尾上机实习 3 的算术练习程序中，将看到函数嵌套调用的实际应用。

7.5 数组作函数参数

前面几节中，我们使用的函数参数均是简单变量，如 int 型、float 型、char 型等。函数参数除了可以是简单变量之外，还可以是数组。数组作函数参数分两种情形，一种是数组元素作函数参数，另一种则是数组名作函数参数。

7.5.1 数组元素作函数参数

数组元素可以作为函数的实参，这种用法与简单变量作函数实参完全相同，这时函数的形参必须是简单变量。

【例 7.10】数组元素作函数参数。

```
main()
{
  int a[3],i,s ;
  for (i=0;i<3;i++)
    scanf("%d",&a[i]) ;
  s=sum(a[0],a[1],a[2]) ;
  printf("s=%d",s) ;
}
sum(x,y,z)
int x,y,z ;
{
  return x+y+z ;
}
```

运行结果如下：

```
12  8  41 ↙
s=61
```

程序中，main()函数调用 sum()函数时使用了数组元素作实参，这时 a[0]、a[1]和 a[2]的值分别传递给了形参 x、y、z。当数组元素作函数实参时，参数的传递也是单向的“值传递”。

7.5.2 数组名作函数参数

数组名作函数参数时，实参和形参都应为数组名，此时，实参与形参的传递为“地址传递”，所谓地址传递是指在调用函数时，系统并没有给形参数组分配新的存储空间，而只是将实参数组的首地址传送给形参数组，使形参数组与实参数组共用同一数组空间。因此，函数中对形参数组的修改，就是对实参数组的修改。

【例 7.11】数组名作函数参数。

```
main()
{
  int a[2],i ;
  void f() ;
  for (i=0 ; i<2 ; i++)
     {
      printf("a[%d]=" , i) ;
      scanf("%d",&a[i]) ;        /* 输入数组 a 中各元素的值 */
     }
  f(a) ;    /* 以数组名 a 作实参调用 f()函数 */
  for (i=0 ; i<2 ;i++)
     printf("a[%d]=%d\n" , i ,a[i]) ;    /* 输出调用 f()函数后数组 a 中各元素的值 */
}

void f(int b[2])
{
  int i;
  for (i=0 ; i<2 ; i++)
     b[i]=b[i]+1;          /* 改变数组 b 中各元素的值 */
}
```

运行结果如下：

```
a[0]=35 ↙
a[1]=21 ↙
a[0]=36
a[1]=22
```

程序中，由于实参数组 a 和形参数组 b 共用一组首地址相同的存储单元，因此，当数组 b 的元素的值发生改变时，数组 a 中对应元素的值也会发生相同的变化。

说明：

1. 用数组名作函数参数时，应该在主调函数和被调函数中分别定义数组。实参数组和形参数组的类型应该一致。

2. 实参数组和形参数组的长度可以一致也可以不一致。

3. 形参数组可不指定大小，在定义形参数组时，在数组名后面跟一个空的方括号。为了在被调函数中处理数组元素的需要，可以另设一个参数，传递数组元素的个数。

如：【例 7.11】可改写如下：

```
main()
{
  int a[2],i ;
  void f() ;
  for (i=0 ; i<2 ; i++)
     {
      printf("a[%d]=" , i) ;
      scanf("%d",&a[i]) ;        /* 输入数组 a 中各元素的值 */
     }
  f(a , 2) ;    /* 以数组名 a 作实参调用 f()函数 */
  for (i=0 ; i<2 ;i++)
     printf("a[%d]=%d\n" , i ,a[i]) ;    /* 输出调用 f()函数后数组 a 中各元素的值 */
```

```
}

void f(int b[] , int n)
{
  int i ;
  for (i=0 ; i<n ; i++)
     b[i]=b[i]+1;          /*  改变数组 b 中各元素的值  */
}
```

7.6 变量的作用域和生存期

7.6.1 变量的作用域

变量的作用域是指变量的有效范围。C 语言允许把一个大的程序分成几个文件，每个文件分别包含若干函数，各个文件可以分别进行编译，然后再连接到一起形成一个完整的可执行文件。程序中各个函数之间的通信可以通过参数传递来实现，也可以通过使用公共的数据来实现。那么，哪些数据可以被各个函数共用而哪些数据又不能共用呢？这就涉及到变量的作用范围问题。

根据变量的作用范围的不同，可将变量分为局部变量和全局变量。

1. 局部变量

局部变量是指在函数内部或程序块内定义的变量。局部变量只在定义它的函数或程序块内有效。在函数内定义的变量以及形式参数均是局部变量。

例如：

```
main()
{
  int x,y;         /* x 和 y 是局部变量，在 main 函数内有效  */
  ……
  {
    int i,j;       /* i 和 j 是局部变量，在复合语句中有效  */
    ……
  }
}

 fun(int a,int b)       /*  形参 a、b 是局部变量，在 fun 函数内有效  */
{
  int m,n;              /* m、n 是局部变量，在 fun 函数内有效  */
  ……
}
```

【例 7.12】理解不同函数中的同名变量。

```
main()
{
  int x,y;
  x=1; y=2;
  f();
```

```
  printf("x=%d，y=%d", x,y);
}

f()
{
  int x,y;
  x=3; y=4;
}
```

运行结果如下：

```
x=1，y=2
```

上例的 main 函数和 f 函数中，分别定义了两组同名的局部变量 x 和 y。由于 main 函数中定义的变量 x、y 只在 main 函数中有效，而 f 函数中定义的变量 x、y 则只在 f 函数中有效，这两组同名变量分别占用两组不同的存储单元，因此，f 函数中对 x、y 的赋值不会改变 main 函数中 x、y 的值。

2. 全局变量

全局变量是指在所有函数之外定义的变量，其作用范围是从定义点开始，直到程序结束。

例如：

```
int x,y;     / * x 和 y  是全局变量，其作用域是从此处开始直到程序末尾  */
main()
{
  ……

}
int a,b;     /* a、b 是全局变量  */
fun1()
{
  ……
}
fun2()
{
  ……
}
```

x、y 的作用范围

a、b 的作用范围

说明：

（1）设置全局变量的目的是为了增加函数间数据联系的渠道，如，当需要从一个函数带回多个返回值时，就可以使用全局变量。

【例 7.13】编写一个函数，求一组学生成绩的总成绩和平均成绩。

分析：

由于 return 语句只能从函数中带回一个返回值，所以不可能让总成绩和平均成绩都靠 return 语句返回。这里，我们可以利用全局变量的特点来解决这个问题，即，使用 return 语句带回一个数据，而另一个数据则通过全局变量来传递。

```
float average ;    /*  全局变量 average 存放平均成绩  */
main()
{
```

```
 int i , num[100] , n , s ;
 printf("输入学生人数： ") ;
 scanf("%d",&n);
 printf("输入%d 个学生的成绩：\n",n);
 for (i=0 ; i<n ; i++)
    {
     scanf("%d" , &num[i]) ;
    }
 s=sum(num,n) ;
 printf("总成绩 = %d\n" , s) ;
 printf("平均成绩 = %.2f\n" , average) ;
 }
 sum(a,m)
 int a[ ] , m ;
 {
  int i , s=0 ;
 for (i=0; i<m ; i++)
    s=s+a[i] ;
 average=(float)s/m ;
 return s ;          /* 返回总成绩 */
}
```

运行结果如下：

```
输入学生人数：5 ↙
输入 5 个学生的成绩：
82 75 86 91 83 ↙
总成绩 = 417
平均成绩 = 83.40
```

这个程序中，总成绩由 sum 函数中的 return 语句返回，而平均成绩则由全局变量 average 带回到主函数中。

（2）全局变量与局部变量可以同名，这时，在局部变量的作用范围内，全局变量不起作用。

【例 7.14】

```
int a ;   /* 定义全局变量 a */
main()
{
 a=10 ;
 f() ;
 printf("a=%d\n",a) ;
}
f()
{
 int a=20 ;        /* 定义局部变量 a */
 printf("a=%d\n",a) ;
}
```

运行结果如下：

```
a=20
a=10
```

这个程序虽然在开始位置定义了全局变量 a，但函数 f()中又定义了局部变量 a，所以，凡是在 f()函数中出现的变量 a 都是指局部变量 a，而不是全局变量 a。一旦 f()函数执行完，该函数中定义的局部变量 a 就立即被释放。

7.6.2　变量的生存期

变量的生存期是指变量存在的时间长短，根据变量生存期的不同，可以将变量分为动态存储变量和静态存储变量。

动态存储是指在程序运行期间根据需要动态分配存储空间的存储方式，即需要时就分配存储空间，不需要时就释放。如形式参数就属于动态存储变量。

静态存储是指在程序运行期间分配固定的存储空间的存储方式。如全局变量就属于静态存储变量。

根据变量的作用域和生存期的不同，可以将变量分为 4 类存储类别，如表 7-1 所示。

表 7-1　变量的存储类别

存储类别	作用域	生存期	存储位置
auto	局部	动态	内存
register	局部	动态	寄存器
static	局部	静态	内存
Extern	全局	静态	内存

1. auto变量

auto 变量即自动变量，这种存储类型是 C 语言程序中使用最广泛的一种类型。C 语言规定，函数内凡未加存储类型说明的变量均视为自动变量，也就是说自动变量可省去说明符 auto。

说明：

（1）自动变量属于局部变量，也就是说，在函数中定义的自动变量，只在该函数内有效，在复合语句中定义的自动变量只在该复合语句中有效。

（2）自动变量属于动态存储方式，只有在定义该变量的函数被调用时才给它分配存储空间，开始它的生存期，函数调用结束时又自动释放存储空间，结束生存期。

（3）由于自动变量的作用域和生存期都局限于定义它的函数或复合语句内，因此，不同的函数和复合语句中可以定义同名的自动变量。

2. register变量

register 变量即寄存器变量。一般情况下，变量的值是存放在内存中的。为了提高程序的执行效率，C 语言允许将局部变量的值放在 CPU 中的寄存器中，这种变量称为“寄存器变量”。通常，可以把一些使用频繁的变量定义为寄存器变量，以加快程序的执行速度。

寄存器变量用关键字 register 标识，其一般形式为：

```
register 类型标识符 变量名 ;
```

说明：

（1）只有非静态的局部变量（包括形式参数）可以作为寄存器变量，而静态的局部变量和全局变量不能作为寄存器变量。例如，下面的定义是错误的：

```
register static int x ;
```

（2）一个计算机系统中的寄存器数目是有限的，因此，不能定义任意多个寄存器变量。

3. static变量

static 变量即静态变量。静态变量可以是局部变量，也可以是全局变量。静态变量的特点是其值始终存在，也就是说，在一次调用到下一次调用之间保留原有的值。

静态变量用关键字 static 说明，其一般形式为：

```
static 类型标识符 变量名 ;
```

说明：

（1）在函数之内定义的静态变量（即局部的静态变量），只能被本函数引用，而不能被其他函数引用，这一点与自动变量相同。但与自动变量不同的是，自动变量在函数每次被调用时进行初始化，而静态变量只在编译阶段初始化一次，在函数执行结束之后，静态变量的值仍然会保留。

【例 7.15】

```
main()
{
 clrscr();
 fun();
 fun();
 fun();
}

fun()
{
 static int x=1;   /* 定义静态的局部变量 x，并初始化为 1 */
 x++;
 printf("x = %d\n",x) ;
}
```

运行结果如下：

```
x = 2
x = 3
x = 4
```

fun()函数中定义的静态局部变量 x，在编译阶段被初始化为 1。第一次调用 fun()函数时，经 x++;语句递增后，x 的值变为 2。第二次调用 fun()函数时，不再对静态局部变量 x 初始化，这时 x 的值为 2，递增后 x 的值为 3。

如果把 fun()函数中的关键字 static 去掉，那么程序的运行结果就会变成：

```
x = 2
x = 2
x = 2
```

（2）在函数外定义的全局的静态变量，可以被各个函数引用。与一般的非静态的全局变量不同的是，静态的全局变量只能在定义它的文件中被访问，而一般全局变量则可以在整个程序的所有文件中被访问。

4. extern变量

extern 变量即外部变量，它是全局变量的另一种提法。外部变量在函数之外定义，它的作用

域是从变量的定义处开始，一直到本程序的末尾。

说明：

（1）外部变量可以被程序中的各个函数所共用。

（2）一个函数可以使用在该函数之后定义的外部变量，这种情况下，必须在该函数中用 extern 说明要使用的外部变量已在函数的外部定义过了，以便让编译程序作出相应的处理。

【例 7.16】

```
main()
{
 extern a ;    /* 说明变量 a 是外部变量 */
 a=10 ;
 printf("a=%d\n",a);
 f();
 printf("a=%d\n",a);
}

int a ;
f()
{
 a=20;
}
```

运行结果如下：

```
a=10
a=20
```

这个程序中，如果把主函数中的 extern a;去掉，那么编译时就会出现下面的出错信息：

```
Undefined symbol 'a' in function main
```

7.7　函数的作用域

一般情况下，函数可以被其他的所有函数调用，即，可以把函数看作是全局的。但如果一个函数被说明成静态的，则该函数只能在定义它的文件中被调用，而其他文件中的函数则不能调用它。

根据函数是否能被其他文件调用，可将函数分为内部函数和外部函数。

7.7.1　内部函数

只能被本文件中的其他函数调用的函数，称为内部函数。定义内部函数的一般形式如下：

```
static 类型标识符 函数名（形参表）
```

如：

```
static float max(float x , float y)
```

7.7.2　外部函数

除了能被本文件中的其他函数调用之外，还可以被其他文件中的函数调用的函数，称为外部函数。定义外部函数的一般形式如下：

```
extern 类型标识符 函数名（形参表）
```

如：

```
extern float max(float x , float y)
```

如果在定义函数时既没有指定为 static，也没有指定为 extern，那么该函数隐含为外部函数。本书前面用到的所有函数都是外部函数。



习题七

1. 填空题

（1）一个 C 程序由主函数和若干_____组成，各个函数在程序中的定义是_____的。

（2）当_____作函数参数时，实参与形参的传递为“地址传递”。

（3）根据变量的作用范围的不同，可将变量分为_____变量和_____变量。根据变量生存期的不同，可以将变量分为_____变量和_____变量。

（4）static 变量的特点是_____。

2. 选择题

（1）如果一个函数有返回值，那么这个函数只有_____个返回值。

A. 1　　B. 2

C. 3　　D. 不确定

（2）下面关于空函数的定义，正确的是_____。

A. int max(int x,int y) ;　　B. int max(int x,int y) {}

C. int max(int x,y) {}　　D. int max(int x; int y) {}

（3）以下错误的描述是_____。

A. 函数调用可以出现在执行语句中。

B. 函数调用可以出现在一个表达式中。

C. 函数调用可以作为一个函数的形参。

D. 函数调用可以作为一个函数的实参。

（4）调用一个不含 return 语句的函数，以下正确的说法是_____。

A. 该函数没有返回值。

B. 该函数返回一个固定的系统默认值。

C. 该函数返回一个用户所希望的函数值。

D. 该函数返回一个不确定的值。

（5）数组名作函数参数时，实参传递给形参的是_____。

A. 数组元素的个数　　B. 数组的首地址

C. 数组第一个元素的值　　D. 数组中所有元素的值

3. 指出并改正下面程序在函数定义或调用中的错误。

（1）main()

```
{
 int a ;
 ……
 f(a);
```

```
    ……
    }
    f(x)
    {……}
```

（2）main()

```
{
  int a ;
  ……
  f(a);
  ……
}
f(x)
float x ;
{……}
```

（3）main()

```
{
  void f() ;
  ……
  m=f() ;
  ……
}
void f()
{……}
```

4. 分析下列程序，写出运行结果。

（1）main()

```
{
  int a=2,b;
  b=f(a);
  printf("b=%d",b);
}
f(x)
int x;
{
  int y;
  y=x*x;
  return y;
}
```

（2）main()

```
{
  int a,b ;
  printf("a=") ;
  scanf("%d",&a) ;
  printf("b=") ;
  scanf("%d",&b) ;
  swap(a,b) ;
  printf("a=%d,b=%d",a,b) ;
}
```

```
swap(x,y)
int x,y ;
{
  int t ;
  t=x ;
  x=y ;
  y=t ;
}
```

（3）

```
int x;
main()
{
  x=1;
  f();
  x++;
  printf("x=%d",x);
}
f()
{
  x++;
}
```

（4）

```
main()
{
  int i , num[5]={1,2,3,4,5};
  f(num);
  for (i=0 ; i<5 ; i++)
     printf("num[%d]=%d\n" , i , num[i]) ;
}
f(a)
int a[ ] ;
{
  int i ;
  for (i=0 ; i<5 ; i++)
     a[i]=a[i]+1 ;
}
```

5. 编程题

（1）编写一个判断奇偶数的函数，要求在主函数中输入一个整数，输出该数是奇数还是偶数的信息。

（2）输入一个以秒为单位的时间值，将其转化成“时：分：秒”的形式输出。将转换操作定义成函数。

（3）编写一个程序，显示如下菜单并实现相应的菜单选择功能：

```
************************************
        1. 求整数 n 的立方
        2. 求整数 n 的立方根
        3. 结束程序
************************************
```

要求：

① 菜单所示的 1 和 2 两项功能分别由两个函数实现。

② 每项功能执行完之后均回到菜单，直到按 3 结束程序的运行。

（4）用递归法求 n!（n!=1*2*3*.....*n）。



上机实习指导

一、学习目标

本章重点介绍了 C 语言中函数的定义和调用方法，以及与函数有关的一些基本概念。

函数是 C 语言程序的基本组成单位。如果要解决的问题很简单，那么程序可能只需要一个主函数就能完成任务，但要解决一个复杂的问题，程序中除了必须有主函数之外，还需要有其他函数共同分担任务。利用函数实现程序的模块化，使得复杂问题得以轻松解决，这是学习本章的根本目的。

通过本章的学习，应达到以下要求：

1. 进一步领会“函数是 C 程序的基本组成单位”这一特点，在实际编程中能利用函数实现程序的模块化。

2. 熟练掌握函数的定义和调用方法。

3. 理解变量的各种存储类别的特点，掌握局部变量和全局变量的使用方法。

4. 理解主调函数与被调函数之间数据传递的特点。

二、主调函数与被调函数之间的数据传递途径

大多数情况下，都需要考虑主调函数与被调函数之间的数据传递。如：调用一个函数时需要参数吗？函数是否有返回值？函数执行完后哪些变量的值发生了变化？等等。

主调函数与被调函数之间的数据传递有以下几种典型情况：

1. 主调函数通过函数参数向被调函数传递数据。

如果是基本数据类型的变量作函数参数，那么数据的传递是单向传递，也称为值传递。这种情况下，数据只能由主调函数传给被调函数。

2. 被调函数通过 return 语句向主调函数传递数据。

函数的值可以通过 return 语句返回，在主调函数中利用函数表达式即可得到函数的返回值。需要注意的是，通过 return 语句，只能得到一个函数的返回值。

3. 利用全局变量实现主调函数与被调函数之间的数据传递。

由于全局变量在整个程序中可以被所有的函数使用，所以，利用全局变量可以很方便地实现主调函数与被调函数之间的数据传递和返回。特别是当你希望从被调函数中得到一个以上的返回值时，就可以使用全局变量。

这里需要强调的一点是，不提倡大量使用全局变量。这是因为结构化程序设计要求各个函数模块之间的联系应尽可能的小，而使用全局变量则会增大各函数模块之间的联系。

4. 利用数组名作函数参数，实现双向数据传递。

数组名作函数参数与基本数据类型作函数参数相比，具有完全不同的特点。C 语言规定，数组名代表数组的首地址，所以，数组名作函数参数时，是将该数组的首地址由实参传递给形参，即，实参数组与形参数组会共用一个相同的数组首地址和一段相同的存储单元。所以，当形参数组元素的值发生改变时，实参数组元素的值也会随着改变。



上机实习一　函数的定义和调用

一、目的要求

1. 熟练掌握函数的定义和调用方法。
2. 实际编程中能灵活运用函数参数和返回值实现函数之间的数据传递。

二、上机内容

1. 上机调试下面的程序，修改其中存在的错误。

（1）main()

```
{
 float a,b,s;
 scanf("%f%f",&a,&b);
 s=sum(a,b);
 printf("sum=%f",s);
}
sum(int x , y)
{
 int s;
 s=x+y;
 return s;
}
```

（2）main()

```
{
 int n;
 printf("n=%d",n);
 printstar(n);
 void printstar(n)
 {
  int i;
  for (i=1;i<=n;i++)
    printf("*");
 }
}
```

2. 运行下列程序，分析并观察运行结果。

（1）main()

```
{
 int x,y,z,t,m;
 scanf("%d,%d,%d",&x,&y,&z);
 t=max(x,y);
 m=max(t,z);
 printf("%d",m);
}
max(a,b)
```

```
    int a,b;
    {
     if (a>b) return(a);
     else return(b);
    }
```

运行时输入：

```
10 , 35 , -20 ↙
```

（2）main()

```
    {
     int a[2];
     printf("a[0]="); scanf("%d",&a[0]);
     printf("a[1]="); scanf("%d",&a[1]);
     s(a);
     printf("a[0]=%d,a[1]=%d", a[0], a[1]);
    }
    s(int b[])
    {
     int t;
     t=b[0];
     b[0]=b[1];
     b[1]=t;
    }
```

3. 完善程序。

下面程序的功能是，输入一个 ASCII 码值，输出从该 ASCII 码开始的连续 10 个字符。在横线处填写正确的语句或表达式，使程序完整。上机调试程序，使程序的运行结果与给出的结果一致。

```
#include <stdio.h>
main()
{
  int ascii;      /*   变量 ascii 存放输入的 ASCII 码值   */
  printf("输入 ASCII 码值：");
  _____ ;
  put(_____);
}
put(n)
_____;
{
  int i,a;
  for (i=1;_____;i++)
     {a=n+i-1;
      putchar(_____);
     }
}
```

运行结果一：

```
    输入ASCII码值：97 ↙
    abcdefghij
```

运行结果二：

```
输入ASCII码值：33 ↙
!"#$%&'()*
```



上机实习二　局部变量和全局变量

一、目的要求

1. 理解局部变量和全局变量的概念及其特点。
2. 能运用全局变量实现函数之间的数据传递。

二、上机内容

1. 上机调试下面的程序，修改其中存在的错误。

（1）main()

```
{
 int a=1,c;
 func();
 c=a+b;
 printf("c=%d",c);
}
int b=2;
func()
{
 b++;
}
```

（2）int a,b;

```
main()
{
 int c;
 a=1;b=2
 c=sum(a,b);
 printf("c=%d",c);
}
sum(a,b)
{
 int s;
 s=a+b;
 return s;
}
```

2. 运行下列程序，分析并观察运行结果。

（1）main()

```
{
 int a;
 a=10;
 f();
 printf("a=%d\n",a);
```

```
}
f()
{
  int a;
  a=20;
  printf("a=%d\n",a);
}
```

（2）int x,y;

```
main()
{
  int n;
  x=1;y=2;
  n=x+y;
  printf("x=%d,y=%d,n=%d\n",x,y,n);
  n=s();
  printf("x=%d,y=%d,n=%d\n",x,y,n);
}
s()
{
  int z;
  x=3;y=4;
  z=x+y;
  return(z);
}
```

（3）int a=10;

```
main()
{
  a-- ;
  f();
  clrscr();
  printf("a=%d\n",a);
}
f()
{
  int a;
  a=15;
}
```

3. 完善程序。

以下程序中，主函数调用了 LineMax 函数，实现在 N 行 M 列的二维数组中，找出每一行上的最大值。

在程序的横线处填写正确的语句或表达式，使程序完整。上机调试程序，使程序的运行结果与给出的结果一致。

```
# define N 3
# define M 4
int max[N];
main()
```

```
{
 int num[N][M]={12,35,2,65,33,68,2,5,1,56,3,10};
 int i;
 _____________________________________;
 for (i=0;i<N;i++)
     printf("第%d 行的最大值是%d\n",i+1,_____________________________);
}
LineMax(int x[N][M])
{
 int i,j;
 for (i=0;i<N;i++)
    {
     max[i]=x[i][0];
     for (j=1;j<M;j++)
        if (x[i][j]>max[i]) _____________________________________;
    }
 }
```

运行结果如下：

```
第 1 行的最大值是 65
第 2 行的最大值是 68
第 3 行的最大值是 56
```



上机实习三　一个应用程序的设计

一、目的要求

1. 认识自顶向下的程序设计方法。
2. 在设计较复杂的程序时，能熟练运用函数实现程序的模块化。
3. 了解随机函数 random()的功能和调用格式。
4. 掌握标准菜单的实现方法以及利用若干个函数分别实现菜单中各个选择项功能的方法。

二、上机内容

在本章开头的 7.1 节中，曾介绍过一个加、减、乘、除算术练习程序。这个上机实习中，我们将给出这个程序的具体要求，以及详细的模块结构图和部分源程序清单。上机试一试，你会发现这个程序既有趣味性，又有实用性。

1. 程序的功能。

我们要设计的是一个帮助小学生进行算术练习的程序，它具有以下几项功能：

（1）提供加、减、乘、除四种基本算术运算的题目，每道运算题中的操作数是随机产生的，且操作数为不超过 2 位数的正整数。

（2）练习者根据显示出的题目输入自己的答案，程序自动判断输入的答案是否正确，并显示出相应的信息。如果练习者的答案错了，程序就发出“报警声”，并给出正确的答案。

（3）用菜单显示提供的四种算术运算。

2. 程序的模块结构。

通常，可以在主函数中实现菜单功能。加、减、乘、除四种算术运算的出题与正误判断则

分别在四个函数模块中实现，这四个模块中有两个功能是共同的，即算术题目的显示和答错题时报警声的产生，因此，这两个共同的功能可以分别用两个函数实现，四个运算模块可调用这两个函数。

整个算术练习程序的模块结构如图 7-4 所示。

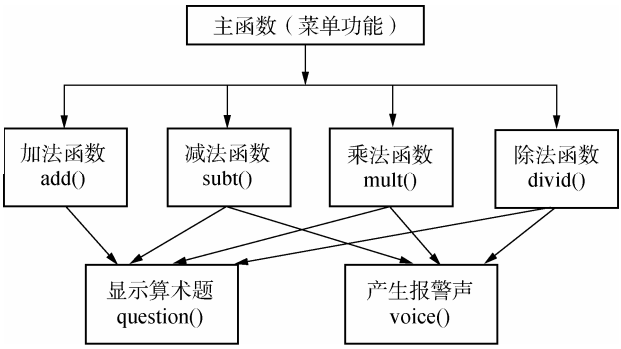


图 7-4　程序的模块结构图

3. 几个细节的考虑。

（1）菜单的实现。

可以用 switch 语句实现菜单的选择。为了方便程序的使用，每项运算功能执行完后，并不立即结束程序，而是又回到菜单，直到选择菜单中的“退出”选项时，才结束程序。这可以通过循环结构实现。

（2）运算题目的随机产生。

可以利用库函数 random()解决随机出题的问题。random()函数的调用格式是：

```
random(int num);
```

其返回值是一个 0 至 num-1 之间的随机数。下面程序段的作用是产生不超过两位数的加法题（操作数的范围是 1 至 99）：

```
num1 = random(99)+1 ;
num2 = random(99)+1 ;
printf("%d + %d = " , num1 , num2) ;
```

4. 部分源程序清单。

程序中共有七个函数，下面只给出 main()、add()、question()和 voice()四个函数的清单，剩下的 subt()、mult()和 divid()三个函数为空函数，其具体内容由你自己完成。要求整个程序上机调试通过。

```
#include <stdlib.h>
#include <time.h>
main()
{
  int x=30,y=6;          /*   x、y 是菜单在屏幕上的起始坐标   */
  char choice;
  randomize();          /*   初始化随机数发生器   */
  while (1)
     {
      clrscr();
      gotoxy(x,y);puts("1.加法练习");              /*   显示菜单   */
      gotoxy(x,y+1);puts("2.减法练习");
```

```
        gotoxy(x,y+2);puts("3.乘法练习");
        gotoxy(x,y+3);puts("4.除法练习");
        gotoxy(x,y+4);puts("5.退       出");
        gotoxy(x,y+6);printf("请选择(1 2 3 4 5):");
        choice=getch();              /*   输入选择项   */
        switch(choice)
           {
            case '1': add();
                      break;
            case '2': subt();
                      break;
            case '3': mult();
                      break;
            case '4': divid();
                      break;
            case '5': exit(0);          /*   结束程序   */
            default : printf("选择有误！按任意键后重新输入……");
                      getch();
           }
      }
}
add()
{
  int i,num1,num2,answer;
  clrscr();
  for(i=1;i<=5;i++)          /*   出 5 道加法题   */
     {
       num1=random(99)+1;
       num2=random(99)+1;
       question(num1,num2,'+');
       scanf("%d",&answer);
       if (answer==num1+num2)
           puts("正确！ ");
       else
          {
           voice();
           printf("错了！ 正确答案是：%d\n",num1+num2);
          }
     }
  printf("\n 加法练习做完了，按任意键返回菜单..... ");
  getch();
}
question(n1,n2,opt)
int n1,n2;
char opt;
{
  printf("%d %c %d =",n1,opt,n2);
```

```
}
voice()
{
  int i;
  for(i=1;i<=3;i++)       /*   连续以三种渐升的频率发声   */
     {
       sound(i*1000);
       delay(2000);
       nosound();
     }
}
/*   自己独立完成以下三个函数   */
subt()
{}
mult()
{}
divid()
{}
```

第8章 文　件

文件是存储在外部存储设备（如磁盘）上的信息集合，一般可分为程序文件和数据文件，本章介绍的文件操作，主要是对磁盘数据文件的使用和操作。

【本章要点】

1. 文件的存储方式与文件指针简介；
2. 文件打开与关闭函数的使用；
3. 文件的读写操作；
4. 随机定位与随机读写函数介绍。

【学习目标】

1. 了解 C 语言处理文件的类型及特点；
2. 掌握文件的打开和关闭方法；
3. 理解文件的读写原理；
4. 能编写简单的文件操作程序。

【课时建议】

讲授 2 课时，上机 2 课时，机动 2 课时。

8.1 C语言文件概述

文件，一般分为程序文件和数据文件。我们已经使用过程序文件了，为了把输入计算机的程序永久保存下来，便把它存放在外部的存储介质上（磁盘、磁带或光盘）。编译好的程序文件，需要时可随时调入内存进行运行；源程序文件，需要时可随时调入内存进行修改和调试，这给用户带来很大的方便。这种存储在外存上的用户程序，就是程序文件的概念。同样，在使用计算机的过程中，大量的计算结果和使用数据（数值的和文字的），也可以象程序那样存储在外部存储介质上，根据需要进行随时存取，这就是数据文件的概念。

本章讲解的是数据文件，主要介绍对磁盘数据文件的使用和操作。使用数据文件至少有三大好处：保存运算的中间结果或最终结果，方便使用；将数据独立于程序之外，便于多处、多人共享；事先录入保存所需数据，可以提高程序运行时的效率。

8.1.1 C语言文件的存储方式及分类

C 语言数据文件存储在磁盘上有两种组织方式，因此，按照存储方式的不同，把 C 语言文件分成两类：一种是按 ASCII 码存储，称作 ASCII 码文件（也叫文本文件）；一种是按二进制码存储，称作二进制文件。

1. ASCII码文件

ASCII 码文件一个字节存放一个 ASCII 码，代表一个字符。其特点是：便于字符的输入/输出处理，但占用空间较大；便于阅读、打印，但与内存中数据进行交换时需要转换，效率较低。

2. 二进制文件

二进制文件存储的都是 0、1 代码，一个字节并不对应一个字符。其特点是：不适于阅读和打印，但占用空间较小；它可将内存中的数据原样输出并保存在文件中，输入/输出不需要进行转换，效率较高。

不管是 ASCII 码文件还是二进制文件，C 语言都将其看作是一个数据流，对文件的存取都是以字节（字符）为单位，它不像其他语言文件一样，有记录的界限，因此称这种文件为流式文件。它允许对文件存取单个字符，这就增加了处理的灵活性。

8.1.2　文件指针

在 C 语言中，对文件操作都是通过标准函数实现的，同时，在使用文件操作函数时，必须定义一个文件指针变量，只有通过文件指针变量，才能找到与其相关的文件，实现对文件的访问。

定义文件指针变量的格式如下:

```
FILE  *fp
```

其中 fp 是用户给出的文件指针变量名，它的类型是 FILE 型。C 语言在标准输入/输出定义文件 stdio.h 中，已经用类型定义语句把流式文件的类型定义为 FILE。FILE 是一个保存文件有关信息（如文件名、文件状态及文件缓冲区位置等）的结构体变量。C 语言规定，使用一个文件就要定义一个文件指针变量，若使用 n 个文件，就要定义 n 个指针变量，使它们指向 n 个文件，就像其他语言中的文件通道号一样，它们在文件读写过程中，可以代表它们指向的文件。

8.2　文件的打开与关闭

要对一个文件进行操作，必须首先打开这个文件，然后才能存取，使用完成之后，还要关闭这个文件，以保证本次操作有效。

8.2.1　文件的打开（fopen（ ）函数）

1. 一般形式

```
<文件指针变量>= fopen(文件名,"方式");
```

例如：fp=fopen("c:\\data\\file.dat","w");

2. 说明

（1）fopen()函数有两个参数，要用逗号隔开；该函数调用后返回一个地址值，通过赋值号“=”赋给<文件指针变量>。

（2）<文件指针变量>（如例中的 fp）要在使用前进行定义，文件打开时，将它指向打开的文件，以便对文件进行读写操作。

（3）“文件名”是字符串或字符串变量，若是字符串变量，应在使用前赋值；给文件名变量赋值或书写文件名字符串时，应注意文件名包括盘符和路径；路径的分隔符要加上转义字符“\”，因此在例中出现的是"c:\\data\\…"；如果被打开的文件在当前盘当前目录下，则盘符和路径可以省略。

（4）“方式”表示欲对所打开的文件进行的访问方式，如例中的“w”，两边的引号不能少。Turbo C 所使用的标识符和含义见表 8-1。

表 8-1 文件打开方式说明

标示符	含义	标示符	含义
r	打开一个 ASCII 码文件只读	r+	打开一个 ASCII 码文件读/写
w	创建一个 ASCII 码文件只写	w+	创建一个 ASCII 码文件读/写
a	打开一个 ASCII 码文件添加	a+	打开一个 ASCII 码文件读/写
rb	打开一个二进制文件只读	rb+	打开一个二进制文件读/写
wb	创造一个二进制文件只写	wb+	创造一个二进制文件读/写
ab	打开一个二进制文件添加	ab+	打开一个二进制文件读/写

当用“w”方式打开一个文件时，若已存在与该文件名相同的文件则会被抹掉，重建一个新文件，若不存在该文件，则新建一个用该文件名命名的文件。

当用“r”方式或“a”方式打开一个文件时，该文件必须存在，不然会返回一个出错信息。

3. 应用举例

【例 8.1】打开一个指定名为“AAA.TXT”的 ASCII 码文件只是为了读，若该文件不存在，返回一个提示信息:

```
#include <stdio.h>
main( )
{
  FILE *fp;       /* 定义文件指针变量 fp */
  If((fp=fopen("AAA.TXT","r"))= =NULL)
  Printf("cannot open this file\n");
  exit(1);
}
```

该例题是一种常用的打开文件的方式，即用 if 语句检查打开指定文件是否存在。如果不存在，则 fopen()函数返回的地址值是 NULL（0 值），屏幕显示“cannot open this file”，告知用户不能打开此文件。exit 函数作用是关闭所有文件，终止正调用的过程。

【例 8.2】从键盘接收一个文件名，为了读写，创造一个二进制文件:

```
#include <stdio.h>
main( )
{
  FILE *fp1;                  /* 定义文件指针变量 fp1 */
  char fn[13];                /* 定义字符数组 fn */
  puts("Enter file name: ");  /* 提示输入文件名 */
  scanf("%s",fn);             /* 接收文件名 */
  fp1=fopen(fn,"wb+");        /* 创造一个二进制文件供读写 */
}
```

8.2.2　文件的关闭（fclose()函数）

1. 一般形式

```
fclose(〈文件指针变量〉);
```

例如：fclose(fp);

2. 说明

（1）〈文件指针变量〉是在文件打开之前定义的，打开后，它始终指向打开的文件；关闭文件，就是使文件指针变量不再指向该文件，同时将尚未写入磁盘的数据（存在内存缓冲区中的数据）写入磁盘，从而保证写入文件的数据完整。

（2）作为良好的习惯，应该在文件操作完毕后及时进行关闭。C 语言文件，在打开时建立一个内存文件缓冲区，读写数据，是通过批处理方式对磁盘进行工作的；当写数据时写满缓冲区才向磁盘写一次，因此，若在缓冲区不满时结束操作，文件中的数据可能不全；但在文件关闭时，不管缓冲区是否已满，都要向磁盘文件中写一次，这样就能保证数据不丢失。

（3）如不注意及时关闭文件，还会造成两个不良的后果：

① 出现文件不够用的错误

C 系统可以提供 20 个文件供用户使用，除了有 5 个被系统的标准文件占用外，用户还可以使用 15 个。看着数量很大，但如果多次打开文件而不注意随时关闭的话，仍然会产生文件不够用的现象。

② 系统自动关闭文件

有时在文件打开太多的时候，系统会自动地关闭一些文件。这样关闭的文件，可能会造成数据的丢失。

8.3　文件的读写

建立和打开文件的目的是为了对其进行读写操作，Turbo C 提供了很丰富的文件读写操作函数，我们在本节只介绍常用的字符、字符串以及格式输入/输出等函数的使用方法，其他函数可参考有关使用手册。

8.3.1　字符的输入和输出

fgetc()和 fputc()是从文件读一个字符和向文件写一个字符的函数，它们的使用方法如下:

1. 向文件中写

```
fputc(ch,fp);
```

其中 ch 是要输入的字符，它可以是字符常量也可以是字符变量；fp 是文件指针变量。fputc()函数若执行成功，返回值是被输出的字符，否则返回值是 EOF（EOF 是符号常量）。

【例 8.3】向 A 驱动器中文件 W83.txt 中，写入一单个字符 X。

```
#include   <stdio.h>      /* 装入原型说明文件*/
main( )
{
 FILE *fp                 /* 定义文件指针变量 fp   */
```

```
    char c='X';              /* 将字符 X 赋予 C    */
    fp=fopen("a:w83.txt","w");
                        /* 在 A 驱动器以只写方式创建并打开 W83.txt 文件 */
    fputc(c,fp);             /* 将 C 的内容字符 X 写入文件 */
    fclose(fp);              /* 关闭文件 */
}
```

程序运行时，屏幕上没有输出信息，输出函数 fputc 将字符 X 写入了磁盘文件 a:w83.txt 中。该文件是文本文件，可以在 DOS 下用 type 命令或在 Windows 系统下打开文件进行验证。

【例 8.4】给文件添加字符

注意

设计该程序时，应考虑到打开已有的文件并向文件中添加字符的打开方式。

```
#include   <stdio.h>
main( )
{
 FILE *fp
 char a='Y',b='Z';              /* a,b 初始化 */
 fp=fopen("a:w83.txt","a");   /* 以追加方式打开文件 */
 fputc(a,fp);                 /* 写入字符 Y */
 fputc(b,fp);                 /* 写入字符 Z */
 fclose(fp);                  /* 关闭文件 */
}
```

请读者考虑，现在的 w83.txt 文件中应该存有什么内容。

2. 从文件中读

```
    Ch=fgetc(fp);
```

其中 ch 是字符变量；fp 是文件指针变量。Fgetc()函数从指定文件中读出一个字符赋给 ch。若执行成功，返回值是被读出的字符，遇到文件尾或出错时返回值是 EOF。

【例 8.5】读出文件 W83.txt 中的字符并显示

注意

设计该程序时，除了应考虑到打开已有的文件并从文件中读取字符的打开方式外，还应考虑如何判断该文件是否存在。

```
#include   <stdio.h>
main( )
{
 FILE *fp;
 If((fp=fopen("a:w83.txt","r"))= =NULL)   /* 以只读方式打开并判断可否 */
  { printf("cannot open this file\n");      /* 显示提示信息*/
    exit(1);}
 char c;                                   /* 定义字符变量 c */
 while(c!=EOF)                             /* 执行 while 循环 */
  {c=fgetc(fp);                            /* 从文件中读出单个字符赋给 c */
   putchar(c);                             /* 显示 C 变量的内容 */
  }
 fclose(fp);                          /* 关闭文件 */
```

```
}
```

该程序运行结果将在屏幕上显示 XYZ。而 EOF 是文件结尾标志，在此作为控制循环的条件。

8.3.2　字符串的输入和输出

fgets()和 fputs()两个函数用来从指定文件中读出一个字符串和写入一个字符串，其常用的格式如下:

1. 从文件中读

```
fgets(<字符串变量>,<字符串长度>,<文件指针变量>);
```

例如：fgets(a,n,fp);

该函数从指定文件中读出一行以'\n'或 EOF 为结尾的字符串，并赋给字符串变量。如果文件中字符串长度大于设定的长度 n，则只读前 n－1 个字符，读出的字符串以'\0'结尾放到字符串变量中。

2. 向文件中写

```
fputs(<字符串变量>,<文件指针变量>);
```

例如：fputs(a,fp);

该函数把给定的字符串写到指定的文件中，并输出一个换行符'\n'。<字符串变量>可以是指向字符串的指针或字符数组名。

【例 8.6】将字符串"Turbo c"写入文件 W86.txt

```
#include   <stdio.h>
main( )
{
 FILE *fi
 char *a='Turbo c';
 fi=fopen("w86.txt","w");
 fputs(a,fi);  /*  向 fi 指出的文件 w86.txt 写入指针变量 a 指向的字符串  */
 fclose(fi);
}
```

【例 8.7】读出 w86.txt 文件中的字符串并在屏幕上显示

```
#include   <stdio.h>
main( )
{
 FILE   *fi;
 char *a;                    /*  存储读出字符串的指针变量说明  */
 if((fi=fopen("w86.txt","r"))==NULL)
   { printf ("cannot open this file\n");
     exit(1);}
 fgets(a,8,fi);              /*  在指定文件中读出长度为 8 个的字符  */
 printf("%s",a);             /*  屏幕显示  */
 fclose(fi);
}
```

该程序运行结果在屏幕上显示字符串 Turbo c。

【例 8.8】将多个字符串 “Turbo c”，“BASIC”，“PASCAL”，“dBASE” 写入文件 W88.txt。

```
#include   <stdio.h>
main( )
{
 FILE *fp;
 static char a[][8]={ "Turbo c","BASIC","PASCAL","dBASE"};
                          /* 数组初始化 */
 int i;
 fp=fopen("w88.txt","w");
 for(i=o; i<=3; i++)
 fputs(a[i],fp);            /* 按 a[0]～a[3]顺序写入文件 */
 fclose(fp);
}
```

请读者自己写出从 w88.txt 文件中读出多个字符串并在屏幕上显示的程序，然后可通过上机实习验证其正确性。

8.3.3 按格式输入和输出

fscanf()和 fprintf()两个函数可以实现对文件按格式读写操作。其使用格式如下:

```
fscanf (<文件指针变量>,<格式控制串>,<参数表列>);
fprintf (<文件指针变量>,<格式控制串>,<参数表列>);
```

其中格式控制串的使用与 scanf()和 printf()中的相同，所不同的是这两个函数的操作对象是文件。

【例 8.9】将字符串“this is a test”和整数 10 与浮点数 20.01 写入文件 W89.txt。

```
#include   <stdio.h>
main( )
{
 FILE   *fp
 If((fp=fopen("w89.txt","wb"))==NULL)
 {printf("cannot open file\n");
  exit(1);}    /* 若磁盘有写保护等打不开文件可提示 */
 fprintf(fp,"this is a test %d%f ",10,20.01);
 fclose(fp);
}
```

【例 8.10】从 w89.txt 文件中读出字符串和数值，并在屏幕上显示:

```
#include   <stdio.h>
main( )
{
 FILE *stream;
 char *a;
 int   b;
 float c;
 *a="            ";
 if(!(stream=fopen("w89.txt","rb"))){
 printf("cannot open file\n");
 exit(1);}                          /* 若文件不存在可提示 */
 fscanf(stream,"%15s",a);           /* 读出字符串 */
 fscanf(stream,"%d%f ",&b,&c，); /* 读出数值 */
```

```
    printf("%15s %d %f\n",a,b,c);          /*  显示  */
    fclose(stream);
  }
```

该程序运行结果显示:

```
    this is a test 10 20.010000
```

注意

fscanf()函数从文件中读取字符串，只当遇到空格符、换行符或文件结束标志 EOF 等时，才结束字符串的读取。

本例能读出并显示该正常结果，是因为【例 8.9】中使用了正确的写入格式。如果将 fprintf（fp，"this is a test %d%f "，10，20.01）；语句换成 fprintf（fp，"this is a test，%d%f "，10，20.01）；请读者考虑本题输出会有什么变化。

【例 8.11】将一批整数写入文件 W811.dat，然后顺序从中读出并显示，直到数据读完为止。

```
  #include   <stdio.h>
  FILE   *fp;
  int a[]= {1,10,50,100,500,1000,1500,23456};
  main( )
  {
   int i,b[1];
   fp=fopen("w811.dat","wb");
   for(i=0; i<=7; I++)                    /*循环写入数据*/
      fprintf("%d",a[i]);
   fclose(fp);
   fp=fopen("w811.dat","rb");
   while(feof(fp)==0)                     /*不是文件尾时读数据*/
    {fscanf(fp,"%d",&b[0]);
     printf("%d",b[0]);
    }
    fclose(fp);
  }
```

该程序运行结果在屏幕上显示:

```
1   10   50   100   500   1000   1500   23456
```

例中使用的 feof()函数是文件尾检测函数，它既适用于二进制文件也适用于文本文件，当文件位置指针已处于文件尾时，它返回一个非 0 值，否则返回 0。前面已经讲过，EOF 可作为文本文件的结束符，但它不能作为二进制文件的结束符。这是因为它在 stdio.h 中定义为−1，是一个合法的二进制整数，字符的 ASCII 码不可能出现−1，因此可以用它作文本文件的结束标志（如【例 8.5】）；而不能用作二进制文件的结束标志。

注意

前三节讲了文件操作的完整过程：打开文件→对文件进行读或写操作→关闭文件。在编写有关文件操作的程序时，初学者容易犯的一个共同的错误是，打开文件的模式和真正操作的不一致性，这一点应该提醒大家特别注意。请看如下一段程序:

……

```
if((fp=fopen("test.dat","r")==NULL)
{ printf("文件 test.dat 打不开！\n");
  exit(1);}
```

```
ch=fgetc(fp);
while(ch!= '#')
{ch+=4;
 fputc(ch,fp);
 ch=fgetc(fp);}
……
```

该段程序，打开文件是以"r"方式——只读方式打开的，而在文件操作时却既要进行读操作，又要进行写操作，显然是不允许的。请读者考虑应如何改正。

8.4 随机文件的读写

在 C 语言中，对于随机文件的读写与有些语言不同，它在 OPEN 时没有特殊的规定，但提供了一些随机定位和读写的函数，可以用于随机文件的读写。

8.4.1 fseek()函数

对流式文件可以进行顺序读写，也可以进行随机读写，关键是如何控制文件的位置指针。文件中有一个位置指针，它总是指向当前的读写位置。 如果位置指针按字节顺序移动，就是顺序读写；如果位置指针可以随意移动位置，就可以实现随机读写。用 fseek()函数就可以实现改变文件的位置指针，其应用的格式为:

```
fseek(<文件指针变量>,<位移量>,<起始点>);
```

其中位移量的类型是长整型，指出从起始点算起，位置指针向前移动多少字节。起始点可用符号常数和数值代码代替，其意义如表 8-2。

表 8-2 文件位置指针起始点

符号常数	数值码	含 意
SEEK_SET	0	从文件头开始
SEEK_CUR	1	从文件指针现行位置开始
SEEK_END	2	从文件的末尾开始

例如:

```
fseek(fp,100L,0);
fseek(fp,100L,SEEK_SET);
```

两个语句都是合法的语句，它的作用是将位置指针移到离文件头 100 个字节处。

8.4.2 fread()与fwrite()函数

这是用于文件读写的另外两个函数，比较适合用于随机文件的读写，其一般使用格式如下:

```
fread(ptr,size,n,fp);
fwrite(ptr,size,n,fp);
```

fread 是从指定的文件（fp）中读 n 个数据块，数据块的长度为 size 个字节，读到 ptr 指针所指的内存地址去。fwrite 是从 ptr 指针所指的内存地址起，读 n 个长度为 size 个字节的数据块写到指定的文件中去。

【例 8.12】将 20 个学生的序号、姓名、分数的数据写入 w812.dat 文件。

```
#include <stdio.h>
```

```
#define NUMB 20
main( )
{
 int I,num;
 chr name[10];
 float score;
 FILE   *fp
 If(!(fp=fopen("w812.dat","wb")))
 {printf("cannot open file \n");
  exit(1);
 }
 for(i=0; i<NUMB; i++)
 {scanf("%d%10s%f ",&num,name,&score);   /* 键盘接收数据 */
  fwrite(&num,2,1,fp);                   /* 写序号整型数 2 字节 */
  fwrite(name,10,1,fp);                  /*写姓名字符串 10 字节 */
  fwrite(&score,sizeof(float) ,1,fp);
                          /* sizeof(float)是测出浮点数长度 */
 }
 fclose(fp);
}
```

【例 8.13】已知 w812.dat 文件中存有 20 个学生的数据，根据序号随机改写某学生的分数。

```
#include   <stdio.h>
main( )
{
 int x;
 long i;
 float score;
 FILE *fp;
 If(!(fp=open("w812.dat","rb+")))
  { printf("cannot open file\n");
    exit(1);
  }
 printf("请输入序号: ");
 scanf("%d",&x);
 i=16*(x−1) +12;     /* 计算序号为 X 的学生的分数在文件中存储位置 */
 printf("请输入新的分数: ");
 scunf(" %f ",&score);
 fseek(fp,I,0);      /* 从文件头开始移动位置指针 i 个字节 */
 fwrite(&score,4,1,fp);      /* 改写分数 */
 fclose(fp);
}
```

请读者思考为什么用“rb+”方式打开文件 ？计算序号为 X 的学生的分数在文件中存储位置的公式为什么为 i=16*(x−1)+12？

【例 8.14】使用 w812.dat 文件，输入序号，读出相应序号的一个学生的数据并显示。

```
#include   <stdio.h>
main( )
{
```

```
    int num，x；
    char name[10]；
    float score；
    long i；
    FILE *fp
    If(!(fp=open("w812.dat","rb")))
      {printf("cannot open file\n")；
       exit(1)；
      }
    printf("请输入序号: ")；
    scanf("%d",&x)；
    i=16*(x−1)；
    fseek(fp,I,SEEk_SET)；
    fread(&num,2,1,fp)；
    fread(name,10,1,fp)；
    fread(&score,4,1,fp)；
    fclose(fp)
    printf("序号: %d 姓名: %s 分数%f ",num,name,score)；
  }
```



习题八

1. 填空

（1）文件是存储在外部存储设备上的_____。一般分为_____文件和_____文件。

（2）C 语言文件存储在磁盘上有两种形式，一种是按_____，另一种是按_____。

（3）定义文件指针变量的格式为_____。

（4）feof（）函数是_____检测函数，当文件位置指针处于_____时，它返回一个____值。

（5）对流式文件可以进行顺序读写，也可以进行随机读写，关键是_____。

2. 回答问题

（1）使用存储在外部存储介质上的数据文件主要有几大好处？是什么？

（2）ASCII 码文件与二进制文件有什么不同？各自的特点是什么？

（3）请解释用 W、r、a、wb、rb+等方式打开文件时代表什么意思？

（4）文件操作完以后应及时关闭，其意义何在？

（5）为什么读二进制文件时使用 feof（）函数检测文件尾，而不用 fgetc（）函数检查 EOF。

3. 程序改错

（1）从键盘输入一些字符，将其逐个写入磁盘文件 xt1.txt 中，直到输入一个"#"为止。

```
  #include   <stdio.h>
  main( )
  {
    FILE *fp；
    If((fp=fopen(" xt.txt ","r"))==NULL)
      {printf("cannot open this file\n")；
       exit(1);}
```

```
    char c;
    while(c=getchar( )!= '#' )
       fgetc(fp);
    fclose(fp);
}
```

（2）设文件 xt2.txt 中存放了一组整数数据，从文件中读并统计其正数、负数和零的个数，在屏幕上显示。

```
#include   <stdio.h>
main( )
{
  FILE   *fp;
  int z=0,f=0,l=0,i=0;
  if((fp=fopen(" xt2.txt","w"))==NULL) /*  打开文件读行吗？ */
     { printf("cannot open this file\n");
       exit(1);}
  while(feof(p)==1)                    /*不是文件尾时读数据并统计*/
     { fscanf(p,"%d",&I);
       if(i>0)z++;
         else if(i=0)f++;
           else l++;
     }
  fclose(p);
  printf("正数 z=%d,负数 f=%d,零 l=%d",z,f,l);
}
```

4. 编写程序

（1）编写将“Turbo c”和“BASIC”写入文件 LX1.txt 的程序。

（2）编写将（1）题的两个字符串从 LX1.txt 文件中读出的程序。

（3）将整数 100，200，300 写入文件 LX2.txt。要求每个数据写完换行。

（4）读出文件 LX2.txt 中的数据并求其和。

（5）将 8，2，6，4，5，9，1，3，7，4 共 10 个整数写入文件 LX3.txt。

（6）使用（5）题中的 LX3.txt 文件，输入数值 i，读出文件中第 i 号数据并显示。



上机实习指导

一、学习目标

本章主要介绍了对磁盘数据文件的使用与操作，包括有关文件的一些基本概念、文件的打开与关闭、文件的读写等。学习本章应达到的要求为：

了解 C 语言处理文件的类型及特点；

掌握文件的打开和关闭方法；

理解文件的读写原理。

二、应注意的问题

1. 打开文件路径时要注意分隔符的书写

用 fopen（）函数打开文件时，可以在文件名参数中指定盘符和路径，但要注意的是路径分

隔符必须是“\\”而不是“\”（因为 C 语言中的转义字符是以“\”开头）。例如，若以只读方式打开 C 盘根目录下 DATA 子目录下的 STU.DAT 文件，应该写成：

```
fp=fopen("c:\\data\\stu.dat","r");
```

2. 执行完文件的读写操作后要及时关闭文件

编写程序时，打开文件这一步骤一般是不会忘掉的，因为不打开文件就无法进行读写操作；而关闭文件却常常会被忽略。不及时关闭文件会引起丢失数据或出现文件不够用的错误信息出现，因此，执行完文件的读写操作后，一定要及时用 fclose()函数关闭文件。



上机实习一　文件的顺序读写

一、目的要求

1. 掌握文件打开与关闭的方法；
2. 弄清楚如何进行文件的顺序读写。

二、上机内容

1. 建立和运行给定程序，分析并观察运行结果

（1）将本章习题中第 3 题的第(1)小题中需要改正的错误改正过来，然后上机调试运行。

运行时输入：

```
abcdefgh ↙
12345 ↙
ABCD# ↙
```

（2）下面程序的功能是，输入一个文件名，显示该文件的内容。

```
#include <stdio.h>
main( )
{
  char ch，*filename；
  FILE *fp；
  Clrscr( )；
  Printf("输入文件名：")；
  Gets(filename)；
  If((fp=fopen(filename，"r"))==NULL)
    {printf("不能打开文件！")；
     exit(1)；
  }
  printf("文件%s 的内容为：n"，filename)；
  while(!feof(fp))；
   {
    ch=fgetc(fp)；
    putchar(ch)；
   }
  fclose(fp)；
}
```

运行结果如下：

```
输入文件名：  xt1.txt ↙
```

文件 xt1，txt 的内容为：

```
abcdefgh
12345
ABCD
```

2. 完善程序

根据程序的功能，在程序的横线处填写正确的语句或表达式，使程序完整。上机调试运行，使结果与题目要求相同。

（1）将输入的一组学生姓名和两门成绩顺序存入磁盘文件中。

```
#include <stdio.h>
main( )
{
 char *filename，*name；
 FILE *fp；
 int i，n，s1，s2；
 clrscr( )；
 printf("输入存盘文件名：")；
 gets(filename)；
 printf("输入学生数：")；
 scanf(" %d ",&n)；
 _____；/*以写入方式打开文件*/
 for(i=1；i<=n；i++)
    {
     printf(" %d: ",i)；
     scanf(" %s%d%d "),name,&s1,&s2)；
     _____；/*将输入的数据写入文件*/
    }
 _____；/*关闭文件*/
}
```

运行结果如下：

```
输入存盘的文件名：student.dat ↙
输入学生人数：4 ↙
1：张三 67 73 ↙
2：李四 86 90 ↙
3：王五 89 91 ↙
```

要求存入文件数据的格式与输入的格式相一致。

（2）输入学生姓名，在文件 student.dat 中查找并显示该学生的姓名及两门成绩。

```
#include <stdio.h>
main( )
{
 char *filename，*name；
 FILE *fp；
 int s1,s2；
 clrscr( )；
 printf("输入读取数据的文件名:")；
 gets(filename)；
 if( _____ )
```

```
    {
      printf("不能打开文件！");
      exit(1);
     }
  printf("输入要查找的学生姓名:");
  gets(stuname);
  f=0;
  while(!feof(fp))
     {
     fscanf( _____,name,&s1,&s2);
     if(strcmp(name，stuname)==___ )
      { printf("%s 的两门成绩如下：\n"，name);
        _____;              /*打印成绩*/
  f=1;
    _____;       /*终止循环*/
      }
   }
  if(f==___ ) printf("找不到该学生的相关数据！");
  fclose(fp);
 }
```

运行结果之一：

```
输入读取数据的文件名：student.dat ↙
输入要查找的学生姓名：李四 ↙
李四的两门成绩如下：
86，90
```

运行结果之二：

```
输入读取数据的文件名：student.dat ↙
输入要查找的学生姓名：赵六 ↙
找不到该学生的相关数据!
```

3. 选做题：将书中【例 8.8】上机运行，并按题后要求写出从文件中读出的程序进行验证。



上机实习二　文件的随机读写

一、目的要求

熟悉文件的位置指针用法及随机文件的读写原理。

二、上机内容

1. 运行程序，分析并观察运行结果

下面程序中所使用的文件 xt1.txt，为本章上机实习一中建立的同名文件，其初始数据为原输入数据。

（1）#include <stdio>

```
main( )
{
  FILE *fp;
  char ch;
```

```
    long n;
    clrscr( );
    n=5;
    if((fp=fopen("xt1.txt","r"))==NULL)
      {printf("不能打开文件");
       exit(1);
      }
    fseek(fp，n，0);
    ch=fgetc(fp);
    putchar(ch);
    fclose(fp);
  }
```

（2）#include <stdio>

```
  main()
  {
   FILE *fp;
   char ch;
   long n;
   clrscr( );
   if((fp=fopen(" xt1.txt","r+"))==NULL)
    { printf("不能打开文件");
      exit(1);
    }
   printf("输入要修改字符的位置:");
   scanf("%ld"，&n);
   printf("输入新的字符:");
   ch=getch( );
   fseek(fp,n−1,0);
   fputc(ch,fp);
   fclose(fp);
  }
```

若输入要修改字符的位置为 5，输入新的字符为 X，该程序运行完后，进入 DOS 状态用 type xt1.txt 命令显示该文件的内容，观察其变化。

2. 完善程序

下面程序的功能是，显示 xt1.txt 文件中从第 n 个字符开始的连续 3 个字符。在程序的横线处填写正确的语句或表达式，使程序完整。上机调试运行，使结果与题目要求相同。

```
#include <stdio>
  main()
  {
   FILE *fp;
   char ch;
   long n;
   int i;
   clrscr();
   if( _____ )
    { printf("不能打开文件");
```

```
        exit(1);
      }
    printf("输入字符位置:");
    scanf("%ld",&n);
    _____________________; /*移动文件的位置指针*/
    for(i=1; i<=3; i++)
      {
       ch=getch();
    printf("%c\n",ch);
      }
    _____________________;      /*关闭文件*/
  }
```

若文件 xt1.txt 的内容为：

```
abcdefgh
12345
ABCD
```

则运行结果如下：

```
输入字符位置：4 ↙
d
e
f
```

第一篇　基础知识模块

第二篇　选用模块

本模块是选用模块，可根据不同地区、不同学制（3 或 4 年）、不同基础、不同办学条件，进行酌情选用。其教学目标为：

1. 了解条件编译的基本原理，并理解宏定义和文件包含处理；

2. 了解指针的概念，理解指向简单变量、字符串、函数的指针；

3. 了解内存动态分配函数的使用，并初步掌握指针在程序设计中的使用方法。

第 9 章　编译预处理

C 语言提供了编译预处理功能，它是对 C 语言的一种扩充。预处理命令以“#”开头，它们可以出现在程序的任何地方，但均在编译之前被处理。编译预处理是一个正文处理程序，它属于 C 语言编译系统的一部分。C 程序中的预处理命令，在 C 编译系统对源程序进行编译之前，先被编译预处理程序“预处理”，然后再将预处理结果和源程序一起被编译系统编译，以得到目标代码文件。编译预处理命令的使用，对提高程序的可读性、可移植性、灵活性及节省程序开发成本都有很重要的意义。

【本章要点】

1. 宏定义的意义和用法；
2. 文件包含和条件编译的使用。

【学习目标】

1. 灵活掌握无参宏和有参宏的使用方法；
2. 掌握文件包含和条件编译的使用。

【课时建议】

讲授 2 课时。

9.1　宏定义

宏定义是用预处理命令#define 实现的，分为带参数的宏定义与不带参数的宏定义两种形式。

9.1.1　不带参数的宏定义

无参宏的宏名后不带参数。一般格式为：

```
#define　宏名　字符串
```

只要是以“#”开头的均为预处理命令。define 为宏定义命令，其中的宏名是一个标识符，字符串可以是常量、关键字、完整的语句或表达式等形式。在编译预处理时，预处理程序把源文件中出现的宏名都用字符串来取代。字符串也可以为空，表示从源文件中删除所定义的宏名。

【例 9.1】

```
#define PI 3.1415926
#define R 3.1
main( )
{
  float s;
  s=2*PI*R;
```

```
    printf("result is %f\n",s);
}
```

经过编译预处理后将得到如下程序：

```
main( )
{
 float s;
 s=2*3.1415926*3.1;
 printf("result is %f\n",s);
}
```

说明：

（1）宏名通常采用大写字母，以便与程序中的其他标识符区别开来。

（2）宏定义是用宏名代替一个字符串，只是作简单的替换，不作语法检查，只有在编译已被宏展开后的源程序时才报错。

（3）字符串可以是一个关键字、某个符号或为空。例如：

```
#define BOOL    int
#define BEGIN {
#define END     }
#define DO
```

（4）一个宏名一旦被定义了，在没有消除该定义之前，它就不能再被定义为其他不同的值。其作用域是从定义的地方开始到该源文件的结束。

（5）#undef 命令可以终止宏定义的作用域。若一个宏名消除了原来的定义，便可被重新定义为其他的值。例如在程序中定义：

```
#define    YES 1
```

后来又用下列宏定义撤消：

```
#undef    YES
```

那么，程序中再出现 YES 时就是未定义的标识符了。也就是说：YES 的作用域是从定义的地方开始到#undef 之前结束。

（6）在进行宏定义时，可以使用已定义过的宏名。例如：

```
#define MESSAGE    "this is a string"
#define PRN    printf(MESSAGE)
```

9.1.2 带参数的宏定义

C 语言允许宏带有参数。宏定义中的参数称为形式参数，宏调用中的参数称为实际参数，对于带参数的宏，在调用过程中，不仅要宏展开，而且要用实参代替形参。

其定义的一般形式为：

```
#define   宏名(形参表)   字符串
```

字符串中应包含有参数表中所指定的形参。例如：

```
#define   SUM(x,y)   x+y
```

在这个宏定义中 x,y 是形式参数。

下面我们来看一个完整的程序：

【例 9.2】

```
#define PI    3.1415926
#define AREA(x)    PI*x*x
main( )
```

```
{
  float r,s;
  r=2.5;
  s=AREA(r);
  printf("%f\n",s);
}
```

对带参数的宏，在进行宏替换时，按#define 命令行中指定的字符串从左向右进行置换。若遇到形参字符就用程序语句中相应的实参来代替。若遇到的字符不是形参则保留原样。上面的程序经预处理后实际上是下面这个程序。

```
main( )
{
  float r,s;
  r=2.5;
  s=3.1415926*r*r;
  printf("%f\n",s);
}
```

说明：

（1）对带参宏进行定义时，宏名与形参表之间不能有空格。否则将空格以后的字符都作为替换字符串的一部分，这样就成了不带参数的宏定义了。例如：

```
#define  S  (x)  PI*x*x
```

则定义的 S 为不带参数的宏名，它代表字符串“(x) PI*x*x”。

（2）要注意用括号将整个宏和各参数全部括起来(用括号完全是为了保险一些)。

【例 9.3】先看用括号将宏及各参数全部括起来，将得到正确结果（如求 27 被 3 的平方除）。

```
#define  AREA(x)  ((x)*(x))
main( )
{
  printf("AREA=%5.1f\n",27/AREA(3.0));
  printf("AREA=%5.1f\n",27/AREA(1.0+2.0));
}
```

运行结果如下：

```
AREA=   3.0
AREA=   3.0
```

【例 9.4】不用括号将宏括起来（如要求 27 被 3 的平方除）。

```
#define  AREA(x)  x*x
main( )
{
  printf("AREA=%5.1f\n",27.0/AREA(3.0));
  printf("AREA=%5.1f\n",27.0/AREA(1.0+2.0));
}
```

运行结果如下：

```
AREA= 27.0
AREA= 31.0
```

原因在于：未使用括号时，第一个 printf 函数输出的是：27.0/3.0*3.0。由于运算符/和*的优先级一样高，结合方向从左到右，故得到结果为 27.0。

第二个 printf 函数输出的是：27.0/1.0+2.0*1.0+2.0。由于运算符/和*的优先级高，所以先计

算/和*，计算完后。再计算 27.0+2.0+2.0，故得到结果为 31.0。

（3）从上面的例题可以看到宏调用与函数调用非常相象。但它们事实上不是一回事。这是需要读者特别注意的一点。分析下面两个例子，看一看它们的区别就明白了。

【例 9.5】利用函数调用输出 1 到 10 的平方。

```
int square (int n)
{
  return(n*n);
}
main( )
{
  int   i=1;
  while (i<=10)
     printf("%d\t",square(i++));
  }
```

运行结果如下：

```
1   4    9    16   25   36   49   64   81   100
```

【例 9.6】利用宏定义对上面程序进行改写。

```
#define square(n)   ((n)*(n))
main( )
{int   i=1;
  while (i<=10)
    printf("%d\t",square(i++));
  }
```

运行结果如下：

```
2          12         30         56         90
```

显然，这不是我们所期望得到的结果。原因在于每次循环时，square(i++)经宏替换后变为：(i++)*(i++)，在输出一个数的平方后，i 增加了 2，所以在输出 2，12，30，56，90 后就结束。


注意

预处理程序用一个字符串代替另一个字符串时，它完全是原封不动地进行替换，不做任何检查。

9.2 文件包含

9.2.1 使用格式

预处理程序中的“文件包含”是指一个源文件可以将另外一个源文件的全部内容包含进来，即将另外的文件包含到本文件之中。包含文件的命令格式有如下两种：

格式 1：#include <filename>

格式 2：#include "filename"

其中 filename 是一个现存的文件，其扩展名一般是“.h”。

格式 1 中使用尖括号< >是通知预处理程序，按系统规定的标准方式检索文件目录。例如，使用系统的 PATH 命令定义了路径，编译程序按此路径查找 filename，一旦找到与该文件名相同的文件，便停止搜索。如果路径中没有定义该文件所在的目录，即使文件存在，系统也将给

出文件不存在的信息，并停止编译。

格式 2 中使用双引号" "是通知预处理程序首先在原来的源文件目录中检索指定的文件 filename；如果查找不到，则按系统指定的标准方式继续查找。

预处理程序在对 C 源程序文件扫描时，如遇到#include 命令，则将指定的 filename 文件内容替换到源文件中的#include 命令行中。

文件包含也是一种模块化程序设计的手段。在程序设计中，可以把一些具有公用性的变量、函数的定义或说明以及宏定义等连接在一起，单独构成一个文件。使用#include 命令把它们包含在所需的程序中。这样也为程序的可移植性、可修改性提供了良好的条件。例如在开发一个应用系统中若定义了许多宏，可以把它们收集到一个单独的头文件中（如：user.h）。假设 user.h 文件中包含有如下内容：

```
#include   <stdio.h>
#include   <string.h>
#include   <malloc.h>
#define   FALSE   0
#define   NO   0
#define   YES   1
#define   TRUE   1
#define   TAB   '\t'
#define   NULL   '\0'
```

当某程序中需要用到上面这些宏定义时，可以在源程序文件中写入包含文件命令：

```
#include   "user.h"
```

9.2.2　使用说明

1. 一个文件包含命令一次只能指定一个被包含文件，若要包含 n 个文件，则要使用 n 个包含文件命令。

2. 在使用文件包含命令时，要注意尖括号<filename>和双引号"filename"两种格式的区别。

3. 文件包含可以嵌套，即在一个被包含文件中又可以包含另一个被包含文件。如文件 user.h 中又使用包含命令将文件 stdio.h、string.h 和 malloc.h 包含进来。

4. 被包含文件（stdio.h、string.h 和 malloc.h）与其所在的包含文件（user.h）在预处理后已成为同一个文件。因此，在使用文件包含命令#include　"user.h" 后，头文件 stdio.h、string.h 和 malloc.h 中的宏定义等内容就在头文件 user.h 中有效，不必再进行定义。

9.3　条件编译

C 语言预处理程序提供了条件编译的功能，可以按不同的条件去编译不同的程序部分，以产生不同的目标代码文件。这对于程序的移植和调试是很有用处的。条件编译有以下 3 种形式，下面分别介绍：

9.3.1　形式 1

格式：#ifdef　标识符

```
    程序段 1
#else
```

```
        程序段 2
    #endif
```

预处理程序扫描到#ifdef 时，判别其后面的标识符是否被定义过（一般用#define 命令定义），从而选择对哪个程序段进行编译。对#ifdef 格式而言，若标识符在编译命令行中已被定义，则条件为真，编译程序段 1；否则，条件为假，编译程序段 2。#else 部分可以省略，若被省略，且标识符在编译命令行中没有被定义时，就没有语句被编译。

【例 9.7】

```
#ifdef  IBM_PC
    #define  INTEGER_SIZE  16
#else
    #define  INTEGER_SIZE  32
#endif
```

若 IBM_PC 在前面已被定义过，如：#define IBM_PC 0

则编译命令行：

```
    #define  INTEGER_SIZE  16
```

否则，编译命令行：

```
    #define  INTEGER_SIZE  32
```

这样，源程序可以不作任何修改就可以用于不同类型的计算机系统。

9.3.2 形式 2

格式：#ifndef 标识符

```
        程序段 1
    #else
        程序段 2
    #endif
```

与形式 1 的区别是将“ifdef”改为“ifndef”。它的功能是，若标识符没有被定义，则条件为真，编译程序段 1；否则，条件为假，编译程序段 2。#else 部分也可以省略。

【例 9.7】若用# ifndef 形式实现，只需改写成下面的形式，其作用完全相同：

【例 9.8】

```
# ifndef  IBM_PC
    #define  INTEGER_SIZE  32
# else
    #define  INTEGER_SIZE  16
# endif
```

9.3.3 形式 3

格式：#if 标识符

```
        程序段 1
    #else
        程序段 2
    #endif
```

预处理程序扫描到#if 时，通过测试表达式值是否为真（非零）来选择对程序段 1 还是程序段 2 进行编译。如果#else 部分被省略，且在表达式值为假时就没有语句被编译。

【例 9.9】

```
#if   X
      printf("|x|=%d",X);
#else
      printf("|x|=%d",-X);
#endif
```

现在先定义 X：#define X　5。运行时，根据 X 的值是否为真（非零），决定对哪一个 printf 函数进行编译，而其他的语句不被编译（不生成代码）。

通过上面的例子可以分析：不用条件编译而直接用条件语句也能达到要求，这样用条件编译有什么好处呢？用条件编译可以减少被编译的语句，从而减少目标代码的长度。当条件编译段较多时，目标代码的长度可以大大减少。

在调试程序时，常常希望输出一些需要的信息，而在调试完成后不再输出这些信息。这可以在源程序中插入如下的条件编译：

```
# ifdef   DO
    printf("a=%d,b=%d\n",a,b);
# endif
```

如果在它的前面定义过标识符“DO”，则在程序运行时输出 a,b 的值，以便在程序调试时进行分析。调试完成后只需将定义标识符“DO”的宏定义命令删除即可。

【例 9.10】输入一个实数，根据需要设置条件编译，使之能以该实数为半径输出圆的面积，或以该实数为边长输出正方形的面积。

```
#define R 1
main()
{
float c,r,s;
printf ("input a number: ");
scanf("%f",&c);
#ifdef   R
      r=3.14159*c*c;
      printf("area of round is: %f\n",r);
#else
      s=c*c;
      printf("area of square is: %f\n",s);
#endif
}
```

本例采用了第一种形式的条件编译。在程序第一行宏定义中，定义了宏 R，因此在条件编译时，要编译#ifdef 后的程序段，故计算并输出圆面积。课后请用其他两种形式改写【例 9.10】，检查运行结果是否相同。



习题九

1. 选择题

（1）将 C 的库函数中数学函数库的头文件包含到程序中，应在程序的头部加上（　　）。

A. #include "stdio.h"　　B. #INCLUDE "stdio.h"

C. #include "math.h"　　D. #define "math.h"

（2）欲将一个文件 file.c 包含到程序中，应该在程序的头部加上（　　）。

A. #include "file.c"　　B. #INCLUDE "FILE.C"

C. include "file.c"　　D. 什么也不加

（3）“文件包含”预处理语句的使用形式中，当#include 后面的文件名用" "括起时，寻找被包含的文件的方式为（　　）。

A. 直接按系统设定的标准方式搜索目录

B. 先在源程序所在目录搜索，再按系统设定的标准方式搜索

C. 仅仅搜索源程序所在的目录

D. 仅仅搜索当前目录

（4）在 C 语言中，编译预处理命令都是以（　　）符号开头的。

A. $　　B. #　　C. &　　D. *

2. 分析下列程序，写出运行结果。

（1）#define　S(x)　x*x

```
main()
{
  int a,k=3;
  a=S(k+1);
  printf(“a=%d\n”,a);
}
```

（2）#include　<stdio.h>

```
#define　MIN(x,y)　(x)<(y)?(x):(y)
main( )
{int i,j,k;
 i=5; j=30;
 k=100*MIN(i,j);
 printf("%d\n", k);
}
```

（3）#define　A　3

```
#define　B　5
#define　PRINT　printf("\n")
#define　PRINT1　printf("%d", A*B); PRINT
#define　PRINT2(x,y)　printf("%d",x*y)
main( )
{PRINT1;
 PRINT2(A+1,B+1);
}
```

（4）#define　N　2

```
#define　M　N+1
#define　NUM　(M+1)*M/2
main()
{int i;
 for(i=1;i<=NUM;i++)
    printf(“%d”,i);
```

```
}
```

3. 编程题

（1）输入两个整数，求它们相除的余数。用带参数的宏来实现。

（2）定义一个交换两个参数值的宏，并写出程序，输入三个数，然后利用宏按从大到小顺序排列输出。

（3）利用条件编译方法编写程序。输入一行字符，使之能将字母全改为大写输出或全改为小写输出。

第 10 章　指　　针

指针是 C 语言中的一个重要的概念，也是 C 语言的一个重要特色。正确而灵活地运用指针可以有效地表示复杂的数据结构，方便地使用字符串、数组和机器语言所能完成的功能，从而可以使程序清晰、简洁，并生成紧凑、有效的代码。对每一个想深入学习 C 语言的人，都应当学习和掌握指针。

【本章要点】

指针的概念、定义和操作；

指针与数组；

指针与函数。

【学习目标】

了解指针的概念并掌握其定义方法；

理解指向变量、字符串、函数的指针；

初步掌握指针与数组结合的使用方法。

【课时建议】

讲授 6 课时，上机 2 课时。

10.1　指针概述

10.1.1　变量的地址与指针变量



图 10-1　不同数据类型变量在内存中占用的空间

我们在编程中定义或说明变量，编译系统就为已定义的变量分配相应的内存单元，也就是说，每个变量在内存中会有固定的位置，即具体的地址。由于变量的数据类型不同，它所占的内存单元数也不相同。

若我们在程序中有如下定义：

```
int a=1, b=2;
float x=3.4, y = 4.5 ;
double m=3.124;
char ch1='a', ch2='b';
```

让我们先看一下编译系统是怎样为变量分配内存的。变量 a，b 是整型变量，在内存中各占 2 个字节；x，y 是实型，各占 4 个字节；m 是双精度实型，占 8 个字节；ch1，ch2 是字符型，各占 1 个字节。由于计算机内存是按字节编址的，设变量的存放从内存 2000 单元开始存放，则编译系统对变量在内存中一种可能的存放情况如图 10-1 所示。

变量在内存中按照数据类型的不同，占内存的大小也不同，都有具体的内存单元地址，如变量 a 在内存的地址是 2000，占据两个字节后，变量 b 的内存地址就为 2002，变量 m 的内存地址为 2012 等。对内存中变量的访问，过去用 scanf("%d%d%f",&a,&b,&x) 表示将数据输入变量的地址所指示的内存单元。那么在访问变量时，首先应找到其在内存的地址，或者说，一个地址唯一指向一个内存变量，我们称这个地址为变量的指针。如果将变量的地址保存在内存的特定区域，用变量来存放这些地址，这样的变量就是指针变量，通过指针对所指向变量的访问，也就是一种对变量的“间接访问”方式。

C 语言规定可以在程序中定义整型变量、实型变量、字符变量等，也可以定义一种特殊的变量，即用于存放地址的变量。例如可以将变量 a 的地址存放在另一个内存单元中，定义变量 a_ptr 为存放整型变量的地址，它被分配为 3000 和 3001 两个字节。可以用下面的语句将 a 的地址存放到 a_ptr 中。

```
a_ptr=&a;
```

这时，a_ptr 的值就是 2000，即变量 a 所占用单元的起始地址。要存取变量 a 的值，可以先找到存放“a 的地址”的单元地址（3000、3001），从中取出 a 的地址（2000），然后到 2000、2001 字节取出 a 的值 1。这种访问方式称为间接访问方式。

通过地址能找到所需的变量单元，因此，地址就是指向了该变量单元，一个变量的地址称为该变量的指针。例如，地址 2000 是变量 a 的指针，地址 2002 是变量 b 的指针。如果用一个变量专门用来存放另一个变量的地址，则称它为指针变量。上面提到的 a_ptr 就是一个指针变量，指针变量的值是变量的地址。

10.1.2　指针变量的定义

指针变量就是存放地址的变量。C 语言规定：在使用变量之前必须先定义。指针变量定义的一般形式为：

```
        类型标识符   *指针变量;
```

例如：int　*p1;

定义了一个指向整型变量的指针变量 p1，即 p1 是一个存放整型变量的地址的变量。

在定义指针变量时要注意以下几点：

1. 一个指针变量必须用符号“*”，表明其后的变量是指针变量。在本例中 p1 是指针变量，而不要误认为“*p1”是指针变量。

2. 定义了一个指针变量 p1 以后，系统就为这个指针变量分配一个存储单元（一般为 2 个字节），用它来存放地址。但此时该指针变量并未指向确定的整型变量，因为该指针变量中并未赋予确定的地址。对无所指向的指针进行操作，会产生不易查到的错误。当指针所指对象不存在时，可以按如下方式为指针赋初值：

```
        int   *p1=NULL;          /*  定义指针变量的同时赋初值  */
    或  int   *p1;   p=NULL;     /*  使用赋值语句给指针变量赋初值  */
```

若要想使指针变量指向一个整型变量，必须将整型变量的地址赋给它。例如：

```
        int   d1,*p1; p1=&d1;
```

赋值语句“p1=&d1;”的作用就是使 p1 指向 d1。

3. 指针变量只能指向同一类型的变量。例如：p1 可以指向一个整型变量，但不能指向一个实型变量。

4. 变量可以定义为指向字符型、实型以及其他类型的变量。例如：

```
        float   *p2;
```

```
char   *ch;
```

10.1.3 指针变量的操作

在 C 语言中有两个有关指针操作的运算符：

1. 地址运算符&。

2. 指针运算符，或间接访问运算符*。

【例 10.1】两个有关指针操作运算符的应用形式举例

```
int   x=1,y=2;
int   *ip;
ip=&x;    /*   ip 现在指向 x     */
y=*ip;    /*   y 现在的值为 1   */
```

说明：

（1）第一行说明语句定义 x，y 是整数类型的变量并赋初值。第二行说明语句定义 ip 是指针变量，注意它还没有指向任何变量（如图 10−2（a））。第三行语句中&x 表示取变量 x 的地址，然后赋给指针变量 ip，即 ip 指向 x。第四行语句中*ip 表示指针变量 ip 所指向的变量，即 x。将*ip（或 x）的值赋给变量 y，所以 y 的值为 1（如图 10-2（b））。

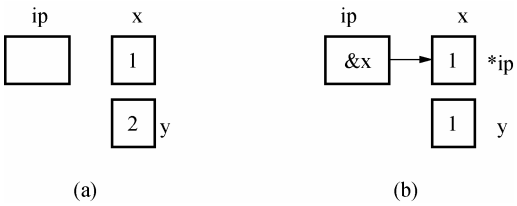


图 10-2

（2）本例中第二行说明语句和第四行语句中均出现*ip，请区分它们的不同含义。int *ip; 表示定义指针变量 ip。它前面的“*”只是表示该变量为指针变量。而 y=*ip 中*ip 代表的是变量，即，指针变量 ip 所指向的变量 x。

（3）第三行“ip=&x”，注意不要写成“*ip=&x”。因为 x 的地址是赋给指针变量 ip 而不是赋给整型变量*ip。

【例 10.2】输入两个整数，按由大到小的顺序输出。

```
main()
{
 int   a1,a2,*p1,*p2,*p;
 scanf("%d,%d",&a1,&a2);
 p1=&a1;p2=&a2;
 if   (a1<a2)
    {p=p1;p1=p2;p2=p;}
 printf("a1=%d,a2=%d\n",a1,a2);
 printf("MAX=%d,MIN=%d\n",*p1,*p2);
}
```

运行情况如下：

```
3,5↙
a1=3,a2=5
MAX=5,MIN=3
```

当通过键盘输入 3,5 时，即 a1=3，a2=5，由于 a1<a2，将 p1 和 p2 交换，交换前的情况如图 10−3（a），交换后的情况如图 10-3（b）。

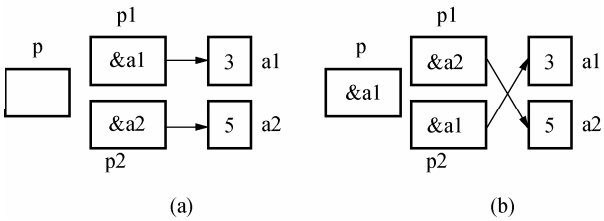


图 10-3

注意 a1 和 a2 的值并未交换，它们仍然保留原值，而 p1 和 p2 的值（即 p1 和 p2 的指向）改变了。程序中指针变量 p 的作用是作为交换 p1 和 p2 的值所用的临时变量。复合语句{p=p1;p1=p2;p2=p;}中指针变量之间赋值的三条语句完成了 p1 和 p2 的值交换的任务。

上例也可通过下面的程序完成。

```
main()
{int   a1,a2,x,*p1,*p2;
 scanf("%d,%d",&a1,&a2);
 p1=&a1;p2=&a2;
 if   (*p1<*p2)
    {x=*p1;*p1=*p2;*p2=x;}
 printf("a1=%d,a2=%d\n",a1,a2);
 printf("MAX=%d,MIN=%d\n",*p1,*p2);
}
```

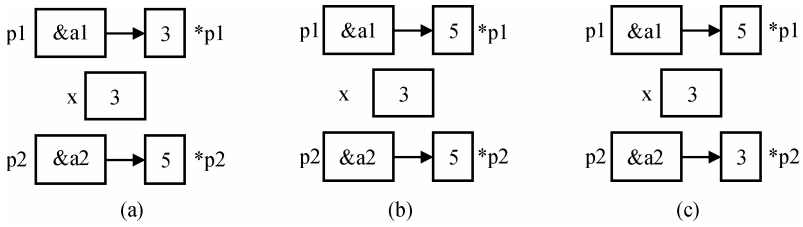


图 10-4

运行情况如下：

```
3,5↙
a1=5,a2=3
MAX=5,MIN=3
```

本例中 p1 和 p2 的指向并未改变，它们仍然保留原值。但 p1 和 p2 所指向的变量的值发生了变化。程序中 x 的作用就是作为交换*p1（指针变量 p1 所指向的变量）和*p2（指针变量 p2 所指向的变量）的值所需的临时变量。复合语句{ x=*p1;*p1=*p2;*p2=x;}中三条赋值语句完成了*p1 和*p2 的值交换的要求。如图 10-4 所示。

通过上面的几个例子，我们可以看出交换指针和交换指针所指向的变量的值有着本质的区别。

10.2 指针与数组

变量在内存存放是有地址的，数组在内存存放也同样具有地址。对数组来说，数组名就是数组在内存存放的首地址。指针变量是用于存放变量的地址，可以指向变量，当然也可存放数组的首地址或数组元素的地址，这就是说，指针变量可以指向数组或数组元素，对数组而言，数组和数组元素的引用，也同样可以使用指针变量。下面就分别介绍指针与不同类型的数组之间的关系。

10.2.1 一维数组的指针

1. 指针对一维数组的表示方法

假设我们定义一个一维数组，该数组在内存会有系统分配的一个存储空间，其数组的名字就是数组在内存的首地址。若再定义一个指针变量，并将数组的首地址传给指针变量，则该指针就指向了这个一维数组。我们说数组名是数组的首地址，也就是数组的指针。而定义的指针变量就是指向该数组的指针变量。对一维数组的引用，既可以用传统的数组元素的下标法，也可使用指针的表示方法。

```
int a[10] , *ptr;        /*  定义数组与指针变量* /
```

做赋值操作：ptr=a；或 ptr=&a[0]；

则 ptr 就得到了数组的首地址。其中，a 是数组名，它代表数组的首地址，&a[0]是数组元素 a[0]的地址，由于 a[0]的地址就是数组的首地址，所以，两条赋值操作效果完全相同。指针变量 ptr 就是指向数组 a 的指针变量。

若 ptr 指向了一维数组，现在看一下 C 规定指针对数组的表示方法：

（1）ptr+n 与 a+n 表示数组元素 a[n]的地址，即&a[n]对整个 a 数组来说，共有 10 个元素，n 的取值为 0～9，则数组元素的地址就可以表示为 ptr+0～ptr+9 或 a+0～a+9，与&a[0]～&a[9]完全一致。

（2）知道了数组元素的地址表示方法，*(ptr + n)和*(a+n）就表示为数组的各元素即等效于 a[n]。

（3）指向数组的指针变量也可用数组的下标形式表示为 ptr[n]，其效果相当于*(ptr+n)。

【例 10.3】从键盘输入 10 个数，以下标法输入/输出数组各元素。

```
# include <stdio.h>
main( )
{
  int n,a[10],*ptr=a;
  for(n = 0;n<=9;n++)
     scanf( "%d" , &a[n]);
  printf("1−−−−−−output! \n");
  for(n=0;n<=9;n++)
     printf("%4d",a[n]);
  printf("\n");
}
```

运行程序：

```
1 2 3 4 5 6 7 8 9 0↙
1−−−−−−output!
```

```
   1   2   3   4   5   6   7   8   9   0
```

【例 10.4】采用指针变量表示的地址法输入/输出数组各元素。

```
# include<stdio.h>
main( )
{
  int n,a[10],*ptr=a;     / *  定义时对指针变量初始化  * /
  for(n=0;n<=9;n++)
     scanf("%d",ptr+n) ;
  printf("2------output! \n") ;
  for(n=0;n<=9;n++)
     printf("%4d",*(ptr+n));
  printf("\n");
}
```

运行结果为：

```
1 2 3 4 5 6 7 8 9 0↙
2------output!
     1   2   3   4   5   6   7   8   9   0
```

2. 指针的运算。

（1）加减算术运算

对于指向数组的指针变量，可以加上或减去一个整数 n。设 pa 是指向数组 a 的指针变量，则 pa+n，pa−n，pa++，++pa，pa−−，−−pa 运算都是合法的。指针变量加或减一个整数 n 的意义是把指针指向的当前位置(指向某数组元素)向前或向后移动 n 个位置。应该注意，数组指针变量向前或向后移动一个位置和地址加 1 或减 1 在概念上是不同的。因为数组可以有不同的类型，各种类型的数组元素所占的字节长度是不同的。如指针变量加 1，即向后移动 1 个位置表示指针变量指向下一个数据元素的首地址。而不是在原地址基础上加 1。

例如：

```
int a[5],*pa;
pa=a;              /*pa 指向数组 a，也是指向 a[0]*/
pa=pa+2;           /*pa 指向 a[2]，即 pa 的值为&pa[2]*/
```

指针变量的加减运算只能对数组指针变量进行，对指向其他类型变量的指针变量作加减运算是毫无意义的。

（2）两个指针变量之间的运算

只有指向同一数组的两个指针变量之间才能进行运算， 否则运算毫无意义。

① 两指针变量相减

两指针变量相减所得之差是两个指针所指数组元素之间相差的元素个数。实际上是两个指针值（地址）相减之差再除以该数组元素的长度（字节数）。例如 pf1 和 pf2 是指向同一浮点数组的两个指针变量，设 pf1 的值为 2010H，pf2 的值为 2000H，而浮点数组每个元素占 4 个字节，所以 pf1−pf2 的结果为(2010H−2000H)/4=4，表示 pf1 和 pf2 之间相差 4 个元素。两个指针变量不能进行加法运算。例如，pf1+pf2 是什么意思呢?毫无实际意义。

② 两指针变量进行关系运算

指向同一数组的两指针变量进行关系运算可表示它们所指数组元素之间的关系。例如：

```
pf1= =pf2 表示 pf1 和 pf2 指向同一数组元素
pf1>pf2 表示 pf1 处于高地址位置
```

pf1<pf2 表示 pf2 处于低地址位置

注意

指针变量还可以与 0 比较。设 p 为指针变量，则 p==0 表明 p 是空指针，它不指向任何变量；p!=0 表示 p 不是空指针。空指针是由对指针变量赋予 0 值而得到的。例如：#define NULL 0 int*p=NULL；对指针变量赋 0 值和不赋值是不同的。指针变量未赋值时，可以是任意值，是不能使用的，否则将造成意外错误。而指针变量赋 0 值后，则可以使用，只是它不指向具体的变量而已。

10.2.2 二维数组的指针

定义一个二维数组：int a[3][4];

表示二维数组有三行四列共 12 个元素，在内存中按行存放，存放形式如图 10.5 所示。其中 a 是二维数组的首地址，可把 a 看作由三个元素 a[0]、a[1]、a[2]组成的一维数组。所以：

a+0 或 a	等价于	&a[0]
a+1	等价于	&a[1]
a+2	等价于	&a[2]
a 或(a+0)	等价于	a[0]
*(a+1)	等价于	a[1]
*(a+2)	等价于	a[2]

也就是*(a+i)等价于 a[i]，a+i 等价于&a[i]。

而每个元素 a[i](0<=i<=2)相当于由四个元素 a[i][0]、a[i][1]、a[i][2]、a[i][3]组成的一维数组。

a[i]+j	等价于	&a[i][j] (0<=j<=3)
*(a[i]+j)	等价于	a[i][j]

所以*(a+i)+j 等价于&a[i][j],*(*(a+i)+j)等价于 a[i][j]。

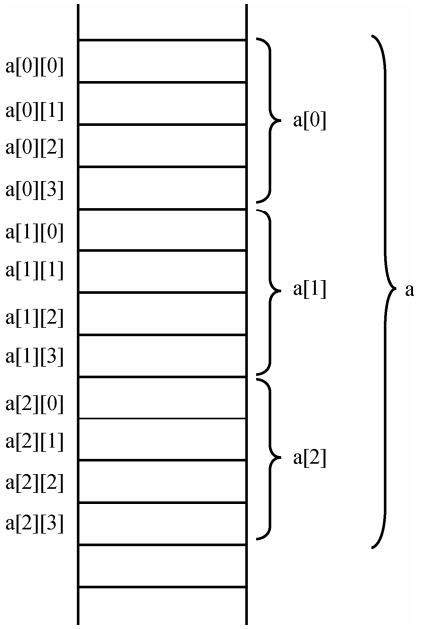


图 10-5 二维数组在内存中的存放形式

我们定义的二维数组其元素类型为整型，每个元素在内存占两个字节，若假定二维数组从 1 000 单元开始存放，则以按行存放的原则，数组元素在内存的存放地址为 1000～1022。用地址法来表示数组各元素的地址。对元素 a[1][2]，& a[1] [2]是其地址，a[1]+2 也是其地址。分析 a[1]+1 与 a[1]+2 的地址关系，它们地址的差并非整数 1，而是一个数组元素的所占字节数 2，原因是每个数组元素占两个字节。对 0 行首地址与 1 行首地址 a 与 a + 1 来说，地址的差同样也并非整数 1，是一行，四个元素占的字节数 8。由于数组元素在内存的连续存放，给指向整型变量的指针传递数组的首地址，则该指针指向二维数组。

```
int *ptr, a[3][4];
```

若赋值：ptr=a[0]；则用 ptr++就能访问数组的各元素。

【例 10.5】用地址法输入/输出二维数组各元素。

```
#include <stdio.h>
main( )
{
```

```
  int a[3][4];
  int i,j;
  for(i=0;i<3;i++)
     for(j=0;j<4;j++)
        scanf("%d" ,a[i]+j) ; / *地址法*/
  for(i=0;i<3;i++)
     {for(j=0;j<4;j++)
        printf("%4d",*(a[i]+j));       /* *(a[i]+j)是地址法所表示的数组元素* /
      printf("\n");
     }
}
```

运行结果为：

```
1 2 3 4 5 6 7 8 9 10 11 12↙
1    2    3    4
5    6    7    8
9   10   11   12
```

【例 10.6】用指针法输入/输出二维数组各元素。

```
# include<stdio.h>
main( )
{
  int a[3][4],*ptr;
  int i,j;
  ptr=a[0];
  for(i=0;i<3;i++)
     for(j=0;j<4;j++)
        scanf("%d",ptr++);       / *指针的表示方法* /
  ptr=a[0];
  for(i=0;i<3;i++)
     {for(j=0;j<4;j++)
         printf("%4d",*ptr++);
       printf("\n");
     }
}
```

运行结果为：

```
1 2 3 4 5 6 7 8 9 10 11 12↙
1    2    3    4
5    6    7    8
9   10   11   12
```

对指针法而言，上述程序中可以把二维数组看作是展开的一维数组：

```
main( )
{int a[3][4],*ptr;
int i,j;
ptr=a[0];
for(i=0;i<3;i++)
   for(j=0;j<4;j++)
      scanf("%d" ,ptr++) ; / *指针的表示方法* /
ptr=a[0];
```

```
for(i=0;i<12；i++)
   printf("%4d" ,*ptr++);
printf("\n");
}
```

运行结果为：

```
1 2 3 4 5 6 7 8 9 10 11 12↙
1   2    3    4    5    6    7    8    9   10   11   12
```

10.2.3 指向字符串的指针

在 C 语言中，字符串是用字符数组来存放的。因此在对字符串操作时，可以定义字符数组，也可以定义字符指针（指向字符型数据的指针）来存取所需的字符。

【例 10.7】利用字符指针访问字符串。

```
#include <stdio.h>
main()
{
 char   str[ ]="A String";
 char   *p;
 p=str;
 while (*p)
    putchar(*p++);
}
```

运行结果：A String

说明：

（1）定义了一个字符型数组 str，并赋初值为字符串：“A String”。如图 10-6 所示。

（2）定义 p 是指向字符型数据的指针变量。

（3）将字符型数组 str 的起始地址赋给指针变量 p，也就是使 p 指向 str[0]。如图 10-7 所示。

（4）用 while 语句输出字符串“A String”。我们知道循环条件表达式的值为零（假）时循环结束。在这里是当*p 等于'\0'（字符串的结束标志）时，循环结束。所以该条件表达式也可以表示成：*p!= '\0'。对于*p++的含义前面已作介绍。

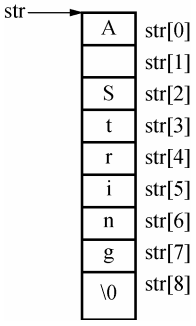


图 10-6

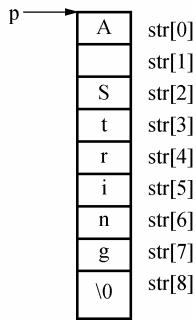


图 10-7

（5）对于程序中的 while 循环也可以用下面的 for 循环来替代：

```
for (;*p!='\0';p++)
   printf("%c\n",*p);
```

在此 printf 函数中用"%c"格式符逐个输出指针变量 p 所指向的字符。p++使 p 指向下一个元

素，如此反复。只要 p 没有指向'\0'，就执行 printf 函数。

（6）输出字符串时也可以用"%s"格式符输出。

```
printf("%s\n",str);
```

或

```
printf("%s\n",p);
```

结果都是输出字符串“A String”。在用格式符"%s"输出时是这样进行的：从给定的地址（str 或 p）开始逐个字符输出。直到遇到'\0'为止。

（7）本例也可以不定义字符数组，而直接定义一个字符型指针变量指向一个字符串。

```
#include   <stdio.h>
main()
{char   *p="A String";
while (*p)
    putchar(*p++);
}
```

程序中虽然没有定义字符数组，但字符串在内存中仍以数组形式存放。它有一个起始地址，占用一片连续的存储单元，以字符'\0 '作为结束标志。char *p="A String";是定义字符型指针变量 p，并使 p 指向字符串“A String”的起始地址。在此读者一定要注意是把字符串的起始地址赋给了 p，而不是将字符串中的字符赋给 p。同样也不能理解为把字符串赋给*p。

【例 10.8】利用一维数组的下标也可以访问字符串。

```
#include   <string.h>
main()
{
 char   *p="A String";
 int   i;
 for (i=0;i<strlen(p);i++)
   prinrf("%c",p[i]);
}
```

运行结果：

```
A String
```

本例中说明 p 是一个字符串指针并赋初值。但在输出语句中并没有使用指针，而是利用了一维数组下标的变化实现了对字符串中字符的访问。道理很简单，无论是字符串还是字符型数组的元素，都是连续存放在内存中的。并且 p[i]和*(p+i)两种引用方法是等价的。

10.3　指针与函数

指针作为 C 语言的重要组成部分，在函数方面的应用非常广泛。本节主要介绍指针作为函数参数；返回指针值的函数以及指向函数的指针变量三个方面的关系。

10.3.1　指针作为函数的参数

在 C 语言中，函数的参数传递有两种方式：传递值和传递地址。前面讲过的整型数据、实型数据或字符型数据等都可以作为函数参数进行传递，这些类型数据传递的是变量的值。那么就称它为“传递值”方式。现在学习了指针变量的概念后，就可以进一步学习用指针变量作为函数参数。我们知道指针变量的值是一个地址，指针变量作函数参数时，传递的是一个指针变

量的值，但这个值是另外一个变量的地址。因此，这种把变量的地址传递给被调用函数的方式称为“传递地址”方式。

【例 10.9】利用指针作函数参数，实现对输入的两个整数交换值后输出。

```
void  swap(int  *px, int  *py)
{
 int  temp;
 temp=*px;
 *px=*py;
 *py=temp;
}
main()
{
 int  x,y;
 int  *p1,*p2;
 scanf("%d,%d",&x,&y);
 printf("x=%d,y=%d\n",x,y);
 p1=&x; p2=&y;
 swap(p1,p2);
 printf("x=%d,y=%d\n",x,y);
}
```

运行情况如下：

```
通过键盘输入 10,20↙
则输出：
x=10,y=20   （第一个 printf 函数输出）
x=20,y=10   （第二个 printf 函数输出）
```

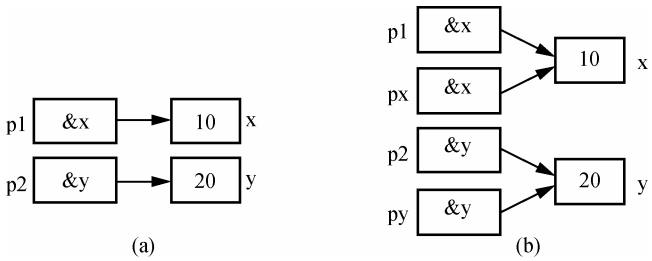
说明：

（1）函数 swap 是用户自定义的函数，它的作用是交换两个指针变量（形参 px 和 py）所指向的变量的值（本例中是 x 和 y 的值）。

（2）主函数中定义了两个整型变量 x 和 y，两个指针变量 p1 和 p2，然后输入 x 和 y 的值（假设输入 10 和 20），并且输出以便与将来交换后再输出形成对比。

（3）赋值语句 p1=&x；和 p2=&y；的作用是使 p1 指向整型变量 x，p2 指向整型变量 y。如图 10−8（a）所示。

（4）在执行 swap 函数时，将实参变量 p1 和 p2 的值（x 和 y 的地址）传递给形参变量 px 和 py（注意 px 和 py 也必须是指向整数类型的指针变量）。这样 px 和 py 的值实际上就是整型变量 x 和 y 的地址，如图 10−8（b）所示。函数执行过程中通过三条赋值语句使*px 和*py 的值互换，也就是使指针变量 px 和 py 所指向的变量 x 和 y 的值互换，如图 10-8（c）所示。函数执行完后，函数中的形参变量 px 和 py 不存在，但变量 x 和 y 仍然存在，如图 10-8（d）所示。



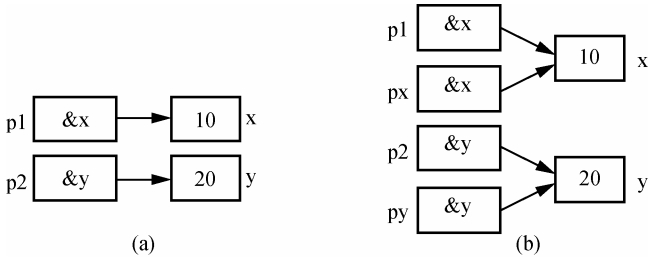


图 10-8

（5）最后 printf 函数输出 x 和 y 的值是交换后的值。

（6）函数 swap 没有返回值，但在主函数中却得到了两个被改变了的变量的值。

【例 10.10】利用指向一维数组元素的指针变量作为函数参数。将数组中各元素按相反顺序输出。

```
void   swap(int   *px , int   *py)
{
 int   x;
 x=*px;
 *px=*py;
 *py=x;
}
main( )
{
 int   i,a[10]={0,1,2,3,4,5,6,7,8,9};
 int   *p1,*p2;
 printf("The original array:\n");
 for (i=0; i<10; i++)
    printf("%d   ",a[i]);
 printf("\n");
 for (i=0; i<5; i++)
    {p1=&a[i];
     p2=&a[10−1−i];
     swap(p1,p2);
     }
 printf("The inverted array:\n");
 for (i=0; i<10; i++)
    printf("%d   ",a[i]);
}
```

运行情况如下：

```
The original array:
0   1   2   3   4   5   6   7   8   9
The inverted array:
9   8   7   6   5   4   3   2   1   0
```

函数 swap 的作用是交换 px 和 py 所指向的两个变量的值。由于在主函数中 p1 和 p2 指向的是两个数组元素，也就是说 p1 和 p2 是两个指向数组元素的指针变量作为实参。swap 函数被调用时，p1 和 p2 传递给了 px 和 py。并在函数中完成了 px 和 py 所指向的变量值的交换，也就是

交换了两个数组元素的值。

前面我们曾经介绍过，数组名代表数组的起始地址。那么数组名也可以作为函数的形参和实参。

上例可改为如下程序完成：

```
void  invert(int  p[ ], int  n)
{
 int  i ,x;
 for (i=0; i<n/2; i++)
 {
   x=p[i];  p[i]=p[n−1−i];  p[n−1−i]=x;
 }
}
main( )
{
 int  i,a[10]={0,1,2,3,4,5,6,7,8,9};
 printf("The original array:\n");
 for (i=0; i<10; i++)
    printf("%d  ",a[i]);
 invert(a,10);
 printf("The inverted array:\n");
 for (i=0; i<10; i++)
    printf("%d  ",a[i]);
}
```

主函数中数组名 a 作为实参将其起始地址传递给被调函数 invert 中的形参 p。在函数 invert 中不具体定义形参数组 p 的大小。数组元素的个数由形参变量 n 传入。这样可以增加函数的灵活性。在这里函数 invert 中的形参数组 p 实际上是一个指针变量，它只有在被调用时，才被赋一个具体的初值。因此本函数说明部分的定义也可以改为 void invert（int *p, int n）。同理，主函数中的实参数组名 a 代表的是数组的起始地址，它是一个地址常数。我们也可以定义一个指针变量指向数组中的某个元素。如：int *pointer1; pointer1=&a[0];（即数组的首地址），主函数中的函数调用语句可改为：invert（pointer1，10）；

总结以上说明，函数的实参和形参均可采用数组名或指针变量。其对应关系如下：

实参	形参
数组名	数组名
数组名	指针变量
指针变量	数组名
指针变量	指针变量

10.3.2 返回指针值的函数

一个函数的类型是由其返回值类型决定的，若函数返回值类型为字符（char）类型，则称它为字符型函数。同理，如果一个函数的返回值为指针，则称它为返回指针值的函数或指针函数；其概念与字符型函数等没有本质区别，只是在函数返回时带回的数值类型是指针类型而已。

返回指针值的函数，一般定义形式为：

```
类型标识符　*函数名(形参表)
{　函数体　}
```

其中“类型标识符”表示函数返回的指针所指向的类型，函数名前的“*”表示此函数的返回值是指针值。

【例 10.11】编写函数，求一维数组中的最大数。

```
int *max(int a[ ],int n)
{
  int *p,i;
  for (p=a, i=1;i<n;i++)
       if (*p<a[i]) p=a+i;
       return(p);
}
main( )
{
  int a[10],*q,i;
  for (i=0;i<10;i++)
       scanf (“%d”,&a[i]);
  q=max(a, 10);
  printf(“\nMAX=%d”,*q);
}
```

运行结果为：

```
34 22 39 55 13 88 32 45 65 67↙
MAX=88
```

说明：函数 max 中定义指针变量 p，使 p 指向数组中的最大数，最后返回指针变量 p 的值（地址）到主调函数中。

10.3.3　指向函数的指针

前面我们学习了指针变量可以指向整型变量、字符串、数组等，现在我们来学习定义一个指针变量指向一个函数。函数在编译时也给分配一片存储区域，它有一个起始地址，既函数的入口地址，这个入口地址称为函数的指针。

【例 10.12】

```
#include　<stdio.h>
int　plus(int　x, int　y)
{
 return(x+y);
}
main( )
{
 int　a=10, b=20, c1, c2;
 int　(*func)( );
 func=plus;
 c1=(*func)(a,b);
 c2=plus(a,b);
 printf("%d+%d=%d\n", a,b,c1);
 printf("%d+%d=%d\n", a,b,c2);
```

```
}
```

运行结果如下：

```
10+20=30
10+20=30
```

说明：

（1）函数 plus 是返回值为整型的函数，func 是一个指向函数（此函数的返回值为整型）的指针变量。注意使用圆括号把*func 括起来，否则 C 编译系统将把它视为返回值为指针（该指针指向整型变量）的函数。

（2）赋值语句“func=plus;”的作用是：将函数的入口地址赋给指针变量 func。如同数组名代表数组起始地址一样，函数名代表函数的入口地址。无需任何括号和参数，更不需要使用地址运算符‘&’。这时 func 就是指向函数 plus 的指针变量。

（3）赋值语句“c1=(*func)(a,b);”中的(*func)就是调用函数 plus。即通过指向函数的指针变量 func 间接调用函数 plus。(*func)后面的实参表必须与 plus 中定义的形参表一致，调用后的返回值是一整型变量值，即 x+y 的值。

（4）程序中分别用函数名和指向函数的指针变量来调用函数 plus，从运行结果来看，两种调用方法是相同的。

（5）指向函数的指针变量只能指向函数的入口处，而不能指向函数中的某一条具体语句。因此象 func++、func- -、func+n 等这样的指针运算是没有意义的。

前面曾讲过函数名代表函数的入口地址，因此函数名可以作为参数传递，被调函数中相应的形参必须说明为指向函数的指针变量类型。

【例 10.13】

```
main( )
{
 int   add(int   x, int   y);
 int   times(int   x, int   y);
 int   sum(int   (*f1)( ), int   (*f2)( ), int   a, int   b);
 int   a=10, b=20, c;
 c=sum(add,times,a,b);
 printf("sum=%d\n", c);
}
int   add(int   x, int   y)
{
 return(x+y);
}
int   times(int   x, int   y)
{
 return(x*y);
}
int   sum(int   (*f1)( ), int   (*f2)( ), int   a, int   b)
{
 return((*f1)(a, b)+(*f2)(a, b));
}
```

运行结果如下：

```
sum=230
```

说明：

（1）在赋值语句“c=sum(add,times,a,b);”中实参 add 和 times 分别与函数 sum 的形参 f1 和 f2 相对应，即将两个实参函数名的首地址传递给函数 sum 的形参 f1 和 f2。

（2）函数 sum 的形参 f1 和 f2 是两个指向函数的指针变量。

（3）函数 sum 利用形参指针变量 f1 和 f2 间接调用函数 add 和 times，间接调用的两个函数的返回值相加后，作为函数 sum 的返回值返回给 main 函数中的变量 c，最后输出变量 c 的值。



习题十

1. 选择题

（1）若已定义“int *p,a;”，则语句“p=&a”中的运算符“&”的含义是（　　）。

A. 位与运算　　B. 逻辑与运算

C. 取指针内容　　D. 取变量地址

（2）下面程序段的运行结果是（　　）。

```
char s[10],*sp=”HELLO”;
strcpy(s,sp);
s[0]='h';s[6]= '! ';
puts(s);
```

A. hELLO　　B. HELLO　　C. hHELLO!　　D. h!

（3）若有定义 int a[3][5],i,j;（且 0≤i＜3,0≤j＜5，则 a[i][j]的地址不正确的是（　　）。

A. &a[i][j]　　B. a[i]+j　　C. *(a+i)+j　　D. *(*(a+i)+j)

（4）若有以下程序：

```
#include <stdio.h>
main(int argc,char *argv[])
{
  while(--argc)
  printf("%s",argv[argc]);
  printf("\n");
}
```

该程序经编译和连接生成可执行文件 S.EXE。

现在 DOS 提示符下键入 S AA BB CC 回车，则输出结果是（　　）。

A. AABBCC　　B. AABBCCS　　C. CCBBAA　　D. CCBBAAS

2. 填空题

（1）以下程序用来输出字符串。

```
#include <stdio.h>
main()
{
  char *a[]={"for", "switch", "if ", "while"};
  char **p;
  for(p=a;p<a+4;p++)
     printf(“%s\n”,_____);
```

```
}
```

（2）exchange 函数的调用语句为“exchange(&a,&b,&c);”。他将 3 个数按由小到大的顺序调整后依次放入 a,b,c 三个变量中，a 中放最大值，请填空。

```
void  swap(int *pt1,int *pt2)
{
 int t;
 t=*pt1;*pt1=*pt2;*pt2=t;
}
void exchange(int *q1,int *q2,int *q3)
{
 if(*q3>*q2)  swap(_____);
 if(*q1<*q3)  swap(_____);
 if(*q1<*q2)  swap(_____);
}
```

（3）以下函数用来求出两整数之和，并通过形参将结果传回，请填空。

```
void func(int x,int y, _____ z)
{ *z=x+y; }
```

（4）下列程序的输出结果是 _____ 。

```
void fun(int *n)
{
while( (*n)--);
   printf("%d",++(*n));
}
main()
{
 int a=100;
 fun(&a);
}
```

3. 分析下列程序，写出运行结果。

（1）

```
main( )
{
 int a=12, b=33, *pa, *pb;
 pa=&a; pb=&b;
 printf("max=%d\n", *pa>*pb?*pa:*pb);
}
```

（2）

```
main( )
 {
  char  s[ ]="ABCDE", *p;
  for (p=s; p<s+5; p++)
     printf("%s\n",p);
 }
```

（3）

```
void  as(int x, int y, int *cp, int *dp)
 {*cp=x+y; *dp=x-y;}
 main( )
 {
  int a=4,b=3,c,d;
  as(a,b,&c,&d);
```

```
        printf("%d,%d\n",c,d);
      }
（4）#include   <stdio.h>
      #include   <string.h>
      void   fun(char   *w, int   m)
      {
       char   s,*p1,*p2;
       p1=w; p2=w+m－1;
       while (p1<p2)
         {s=*p1; *p1=*p2; *p2=s; p1++; p2－－;}
       }
      main( )
      {
       char   a[ ]="ABCDEFG";
       fun(a, strlen(a));
       puts(a);
      }
```

4. 编程题

（1）输入三个数，要求设三个指针变量 p1，p2，p3，使 p1 指向三个数的最小数，p2 指向中间数，p3 指向最大数，然后按由小到大的顺序输出。

（2）写一个函数，使 n 个整数按由小到大的顺序排列。要求在主函数中输入 10 个数，并输出排好序的数。

（3）通过键盘输入一行字符，统计其中大写字母和小写字母的个数（要求利用函数来实现）。

（4）编一个函数 index，从一个字符串 str 中寻找一个字符 ch 第一次出现的位置（返回地址形式）。如果在字符串中找不到指定的字符，则返回 0（提示：此函数定义形式为：char *index(char *str, char ch)）。



上机实习指导

一、学习目标

本章重点介绍了指针的基本概念和初步应用。应该说，指针是 C 语言的重点，也是 C 语言的特色。但是指针使用太灵活，运用不当可能会使整个程序遭受破坏。因此，要求读者使用指针要特别小心，多上机调试程序，以便能灵活应用。通过本章学习，应达到以下要求：

1. 了解指针的概念、指针变量定义、取地址运算符&和指针运算符*的应用。

2. 理解指向简单变量、字符串、函数的指针。

3. 理解指针变量与简单变量在用法上的区别，初步掌握指针在编程中的使用方法。

二、应注意的问题

1. 关于指针变量的定义

指针变量不同于整型、实型、字符型和其他类型的变量，它是专门存放地址的。例如：int　*p; 定义 p 是指针变量，千万不要把*p 理解为指针变量。*p 是指针变量所指向的变量，它是一个整型变量。

2. 关于数组名作为函数的形式参数

在函数的形式参数中可以使用数组名作为参数。但这时的数组名实际上是一个指针变量，系统并不为这个形参数组分配具体存储单元，而是与实参数组共用一片空间。



上机实习　指针的简单应用

一、目的要求

1. 了解指针变量的定义及取地址运算符&和指针运算符*的应用；
2. 初步掌握指向一维数组的指针变量定义及应用；
3. 初步掌握指针变量作为函数参数的传递方式。

二、上机内容

1. 运行程序，分析并观察运行结果。

（1）main()

```
{
 int *p,a;
 clrscr( );
 scanf("%d",&a);
 p=&a;
 printf("%d",*p);
}
```

（2）main()

```
{
 char   *p, *s="abcdef ";
 p=s;
 while (*p!='\0')   p++;
 printf("%d\n", p-s);
}
```

（3）func(t)

```
int *t ;
{
 *t=10 ;
}
main()
{
 int *p,a;
 clrscr();
 a=2;
 p=&a;
 func(p);
    printf("a=%d",a);
}
```

（4）int ff (int b[], int n)

```
{
 int   i, r;
 r =1;
 for (i=0; i<n; i++)
```

```
    r =r*b[i];
  return(r);
}
main( )
{
  int    x, a[ ]={2,3,4,5,6,7,8,9};
  x=ff(a,3);
  printf("%d\n" , x);
}
```

2. 完善程序

请在横线处填写正确的表达式或语句，使程序完整。上机调试程序，使之能将一个字符串的内容颠倒过来。

```
#include <stdio.h>
#include <string.h>
void inv(char    str[ ])
{
  int    i,j,_____;
  for (i=0,j=strlen(str)_____ ; i<j; i++, j--)
    {k=str[i]; str[i]=str[j];  _____}
}
main( )
{
  char    *p;
  gets(p);
  inv(p);
  puts(p);
}
```

附录A　ASCII码字符表



ASCII 码	字符	控制字符	ASCII 码	字符	ASCII 码	字符	ASCII 码	字符
0000000	(null)	NUL	0100000	(space)	1000000	@	1100000	‘
0000001	☺	SHO	0100001	!	1000001	A	1100001	a
0000010	☻	STX	0100010	”	1000010	B	1100010	b
0000011	♥	ETX	0100011	#	1000011	C	1100011	c
0000100	♦	EOT	0100100	$	1000100	D	1100100	d
0000101	♣	ENQ	0100101	%	1000101	E	1100101	e
0000110	♠	ACK	0100110	&	1000110	F	1100110	f
0000111	(beep)	BEL	0100111	’	1000111	G	1100111	g
0001000	■	BS	0101000	(	1001000	H	1101000	h
0001001	(tab)	HT	0101001	)	1001001	I	1101001	I
0001010	(line feed)	LF	0101010	*	1001010	J	1101010	j
0001011	(home)	VT	0101011	+	1001011	K	1101011	k
0001100	(form feed)	FF	0101100	,	1001100	L	1101100	l
0001101	(carriage return)	CR	0101101	-	1001101	M	1101101	m
0001110	♫	SO	0101110	.	1001110	N	1101110	n
0001111	☼	SI	0101111	/	1001111	O	1101111	o
0010000	►	DLE	0110000	0	1010000	P	1110000	p
0010001	◄	DC1	0110001	1	1010001	Q	1110001	q
0010010	↕	DC2	0110010	2	1010010	R	1110010	r
0010011	!!	DC3	0110011	3	1010011	S	1110011	s
0010100	¶	DC4	0110100	4	1010100	T	1110100	t
0010101	§	NAK	0110101	5	1010101	U	1110101	u
0010110	▬	SYN	0110110	6	1010110	V	1110110	v
0010111	↨	ETB	0110111	7	1010111	W	1110111	w
0011000	↑	CAN	0111000	8	1011000	X	1111000	x
0011001	↓	EM	0111001	9	1011001	Y	1111001	y
0011010	→	SUB	0111010	:	1011010	Z	1111010	z
0011011	←	ESC	0111011	;	1011011	[	1111011	{
0011100	∟	FS	0111100	<	1011100	\	1111100	\|
0011101	◆	GS	0111101	=	1011101	]	1111101	}
0011110	▲	RS	0111110	>	1011110	^	1111110	~
0011111	▼	US	0111111	?	1011111	—	1111111	⌂

附录B　Turbo C库函数简介

Turbo C 的库函数是十分丰富的，其 2.0 版本提供了 300 多个函数，它涉及到代码分类、目录路径管理、过程处理、字转换、输入/输出、环境接口、内存分配与操作、数学运算、日期与时间、图形图象等几十大类。为了节省篇幅，同时根据本层次的教学需要出发，本附录只列出一些最基本的、常用的函数。

一、数学函数（应使用头文件math.h）

1. double sin(double x);
返回 Sinx。
2. double asin(double x);
返回 Arcsinx。
3. double cos(double x);
返回 Cosx。
4. double acos(double x);
返回 Arccosx。
5. double tan(double x);
返回 tanx。
6. double atan(double x);
返回 Arctanx。
7. double ceil(double x);
返回最小的不小于 x 的整数。
8. double floor(double x);
返回最大的不大于 x 的整数。
9. double fabs(double x);
返回 double 浮点数的绝对值。
10. double exp(double x);
返回 e^x 的值。
11. double pow(double x,double y);
返回 x^y 的值。
12. double pow10(int p);
返回 10^p 的值。
13. double log(double x);
返回 Lnx 的值。

14. double log10(double x);

返回 $Log_{10}x$ 的值。

15. double fmod(double x,double y);

返回 x/y 的值。

16. double sqrt(double x);

返回 x 开平方的值。

★以下四个数学函数应使用头文件 stdio.h

17. int abs(int i);

返回 i 的绝对值。

18. long labs(long n);

返回 n 的绝对值。

19. int rand(void);

返回 0～215−1 的伪随机数。

20. void srand(unsigned seed);

用来设置由 rand 所产生的伪随机数的起始点。

二、字符转换函数

(一) 数字与字符串转换函数（应使用头文件stdlib.h）

1. double atof(char *str);

把由 str 所指向的字符串转换为一个双精度数。

2. int atoi(char *str);

把由 str 所指向的字符串转换为整型数。

3. long atol(char *str);

把由 str 所指向的字符串转换为长整型数。

4. char *ecvt(double value,int ndigrt,int *decpt,int *sign);

5. char *fcvt(double value,int ndigrt,int *decpt,int *sign);

6. char *gcvt(double value,int ndigrt,char *buf);

以上三个函数都是把浮点数 value 转换成长度为 ndigrt 字符串，返回一个指向这个字符串的指针。从串开始到小数点的位置存在 decpt, 若结果为负,则 sign 所指的值为非 0。在 gcvt 函数中，将串存在 buf 中。

7. char *itoa(int num,char *str,int radix);

8. char *ltoa(long num,char *str,int radix);

9. char *ultoa(unsigned long num,char *str,int radix);

以上三个函数分别将整数、长整数、无符号长整数按 radix 规定的进制转换成字符串存入 str 中。

(二) 字符判断与大小写转换函数（应使用头文件ctype.h）

10. int isalpha(int ch);

判断 ch 是否为字母,返回 1(是)或 0(不是)。

11. int islower(int ch);

判断 ch 是否小写字母,返回 1(是)或 0(不是)。

12. int isupper(int ch);
判断 ch 是否大写字母,返回 1(是)或 0(不是)。
13. int tolower(int ch);
如果 ch 是个字母,返回 ch 的小写体字母。
14. int toupper(int ch);
如果 ch 是个字母,返回 ch 的大写体字母。
★以下两个函数应使用头文件 string.h
15. char *strlwr(char *str);
将 str 串中大写字母变成小写字母,其余不变。
16. char *strupr(char *str);
将 str 串中小写字母变成大写字母,其余不变。

三、字符串操作函数(应使用头文件string.h)

1. char *strchr(char *str,char ch);
在 str 串中从前往后找字符 ch,返回首次出现 ch 的位置指针,没找到返回 NULL。
2. char *strstr(char *str1,char *str2);
在 str1 中找含有 str2 的位置,返回第一个位置,若无返回 NULL。
3. int strcmp(char *str1,char *str2);
比较 str1 和 str2，当(str1<str2),返回值<0；(str1=str2),返回值=0；(str1>str2),返回值>0。
4. char *strcpy(char *str1,char *str2);
把 str2 指向的字符串复制到 str1 中去,返回 str1 指针。
5. char *strcat(char *str1,char *str2);
将 str2 串接到 str1 串后面。
6. unsigned *strlen(char *str);
返回 str 串的字符长度,不计'\0'结尾标志。

四、输入/输出函数

(一) 说明在stdio.h文件中的输入/输出函数

1. int fclose(FILE *fp);
关闭文件 fp,释放缓冲区。
2. int fcloseall(void);
关闭所有打开的文件。
3. int feof(FILE *fp);
测试文件是否遇文件结束符 EOF。
4. FILE *fopen(char *fname,char *mode);
按 mode 规定的操作方式打开 fname 指向的文件。
5. int getc(FILE *fp);
6. int fgetc(FILE *fp);
以上两个函数都是从 fp 指定的文件中获取一个字符作为返回值。
7. char *fgets(char *str,int n,FILE *fp);
从 fp 指定的文件中获取 n－1 个字符放在 str 中。

8. int putc(int ch,FILE *fp);
9. int fputc(int ch,FILE *fp);
以上两个函数都是输出一个字符到 fp 指向的文件中。
10. int fputs(char *str,FILE *fp);
输出字符串 str 到 fp 指向的文件中。
11. int fprintf(FILE *fp,char *format,arg_list);
将参数表 arg_list 中各参数按格式控制串 format 的指定格式输出到 fp 指向的文件中去。
12. int fscanf(FILE *fp,char *format,arg_list);
从 fp 指定的文件中按 format 给定格式将输入数据送到 arg_list 所指向的内存单元。
13. int fseek(FILE *fp,long offset,int origin);
将 fp 指定的文件位置指针移到以 origin 为起始点,以 offset 为偏移量的位置去。
14. long ftell(FILE *fp);
返回 fp 指向的文件位置指针的当前位置。
15. int rewind(FILE *fp);
将文件位置指针返回到文件的开始处。
16. int fread(void *buf,int size,int count,FILE *fp);
从文件 fp 中读取长度为 size 的 count 个数据项放到 buf 指向的内存区。
17. int fwrite(void *buf,int size,int count,FILE *fp);
将 buf 指向的内存区中长度为 size 的 count 个数据写入 fp 文件中去。
18. int getchar(void);
从标准输入设备中读取一个字符。
19. char *gets(char *str);
从标准输入设备中读入一个字符串。
20. int printf(char *format,arg_list);
将参数表 arg_list 的值按 format 的格式输出到标准输出设备。
21. int putchar(int ch);
输出字符 ch 到标准输出设备。
22. int puts(char *str);
输出字符串 str 到标准输出设备。
23. int scanf(char *format,arg_list);
从标准输入设备中按 format 指定格式输入数据送到 arg_list 所指向的内存单元。

(二) 说明在io.h文件中的输入/输出函数

24. int close(int fd);
关闭 fd 文件。
25. int creat(char *filename,int pmode);
以 pmode 指定方式建立一个新文件 filename。
26. int eof(int fd);
测试 fd 文件是否到了文件尾(EOF)。
27. long filelength(int fd);
返回 fd 文件的字节长度。

28. int open(char *filename,int access,int mode);
以 access 访问模式打开名为 filename 的文件,当建立新文件时用 mode 设置操作方式。
29. int read(int fd,void *buf,int nbyte);
从 fd 指向的文件中读取 nbyte 字节存入 buf 中。
30. int write(int fd,void *buf,int nbyte);
把 nbyte 个字节从 buf 缓冲区中写到 fd 文件中去。

(三) 说明在conio.h文件中的输入/输出函数

这一部分函数都是用于文本窗口的,受文本窗口的控制。
31. void window(int left,int top,int right,int bottom);
将所规定的屏幕上的长方形为文本窗口。
32. void clrscr(void);
清除全部文本窗口。
33. void clreol(void);
从光标处清除文本窗口一直到窗口结尾。
34. void delline(void);
清除光标所在行。
35. void gotoxy(int x,int y);
设定光标到指定的(x,y)位置。
36. void highvideo(void);
选择高亮度显示方式。
37. void lowvideo(void);
选择低亮度显示方式。
38. void normvideo(void);
选择正常亮度显示方式。
39. void textbackground(int color);
设置文本窗口字符的背景颜色。
40. void textcolor(int color);
设置文本窗口字符的颜色。
41. int cputs(const char *str);
写一字符串到屏幕,受文本窗口约束。
42. int getch(void);
返回从键盘读入的字符,不显示出来。
43. int getche(void);
返回从键盘读入的字符,并显示出来。
44. int putch(int ch);
向屏幕输出一字符 ch。

五、图形函数（应使用头文件graphics.h）

1. void far closegraph(void);
关闭图形模式。

2. void far detectgraph(int far *driver,int far *mode);

检查硬件配置,确定所用的图形驱动程序和模式。

3. void far initgraph(int far *driver,int far mode,char far *path);

从 path 指定的路径装入图形的驱动程序并初始化图形系统。

4. void far restorecrtmode(void);

恢复屏幕模式在图形初始化前的模式。

5. void far setgraphmode(int mode);

设置系统图形模式并清屏幕。

6. int far getpixel(int x,int y);

获得点(x,y)的颜色。

7. void far putpixel(int x,int y,int color);

在点(x,y)处画一个规定颜色的点。

8. int far getmaxx();

返回屏幕 x 的最大值。

9. int far getmaxy();

返回屏幕 y 的最大值。

10. void far setbkcolor(int color);

设置现行背景颜色。

11. void far setcolor(int color);

设置现行作图颜色。

12. void far arc(int x,int y,int stangle,int endangle,int radius);

画圆弧线,圆心(x,y),半径 radius,开始和结束角度为 stangle 和 endangle。

13. void far circle(int x,int y,int radius);

画圆,圆心为(x,y),半径为 radius。

14. void far ellipse(int x,int y,int stangle,int endangle,int xradius,int yradius);

画椭圆弧线,圆心(x,y),开始和结束角度为 stangle 和 endangle,横轴和纵轴半径为 xradius 和 yradius。

15. void far line(int x0,int y0,int x1,int y1);

画一条直线从(x0,y0)到(x1,y1)。

16. void far lineto(int x,int y);

从现行坐标点画线到(x,y)。

17. void far linerel(int dx,int dy);

从现行坐标点按相对增量画线到(x+dx,y+dy)。

18. void far moveto(int x,int y);

将画笔抬起移动到点(x,y)。

19. void far moverel(int dx,int dy);

将画笔从现行位置移动到点(x+dx,y+dy)。

20. void far rectangle(int left,int top,int right,int bottom);

按左上角和右下角坐标点画长方形。

21. void far drawpoly(int numpoints,int far *shape);

给出顶点数 numpoints 和顶点坐标数组 shape,画一多边形。

22. void far setlinestyle(int style,unsigned pattern,int width);

设置线的类型及宽度。

23. void far setfillstyle(int pattern,int color);

设置现行填充图模和颜色。

24. void far bar(int left,int top,int right,int bottom);

按现行图模和颜色填充给定长方形。

25. void far bar3d(int left,int top,int right,int bottom,int depth,int topflag);

按现行图模和颜色填充一个三维棒图。

26. void far fillpoly(int numpoints,int far *points);

画并填充一个多边形，点数为 numpoints，坐标在 points 指向的数组中，填充为现行图模和颜色。

27. void far floodfill(int x,int y,int bordercolor);

按现行图模和颜色填充封闭的区域，(x,y)为区域内的坐标点，bordercolor 是封闭区域线的颜色。

28. void far pieslice(int x,int y,int stangle,int endangle,int radius);

画并填充一个扇形，参数意义同 12。

29. void far sector(int x,int y,int stangle,int endangle,int xradius, int yradius);

30. void far cleardevice(void);

清除图形屏幕。

31. void far settextjustify(int horiz,int vert);

设置图形模式下文本输出水平和垂直方位的输出方式。

32. void far settexttyle(int font,int direction,int charsize);

设置文本各种特性。

33. void far outtext(char far *str);

在现行位置输出一串字符。

34. void far outtextxy(int x, int y,char far *str);

在指定的(x,y)位置输出一串字符。

六、其他函数

(一) 处理目录和路径的函数（应使用头文件dir.h）

1. int setdisk(int drive);

设置现行磁盘为指定盘（0:A,1:B,2:C,...）。

2. int chdir(char *path);

改变工作目录为 path 规定的目录（如：”A:\\tc”）。

3. int mkdir(char *pathname);

创造一个按 pathname 给定的目录。

4. int rmdir(char *pathname);

删去一个按 pathname 指定的目录。

(二) 过程处理函数（应使用头文件process.h）

5. void abort(void);

非正常地终止程序。

6. void exit(int status);

终止程序,退出之前关闭所有的文件,完成缓冲输出。

7. int system(char *command);

调用 MS-DOS 命令 command。

(三) 内存分配函数（应使用头文件alloc.h）

8. void *calloc(unsigned num,uttime(struct time *t);

获得 DOS 当前系统的时间填入 t 指向的结构 time 中。

9. void *malloc(unsigned size);

同 8，其内存的大小为 size。

10. void free(void *ptr);

释放以前分配的内存块，ptr 指向这个块的第一个字节。

11. void *realloc(void *ptr,unsigned newsize);

调整已分配的内存块大小到 newsize。

(四) 有关日期和时间的函数（应使用头文件dos.h）

12. void getdate(struct date *d);

获得 DOS 当前日期，填入 d 指向的结构 date 中。

13. void gettime(struct time *t);

获得 DOS 当前时间，填入 t 指向的结构 time 中。

14. void setdate(struct date *d);

15. void settime(struct time *t);

以上两个函数，分别按 d,t 指向的结构中的数值设置当前系统的日期和时间。

附录C　Turbo C编译错误信息

Turbo C 编译程序的诊断错误分三种，即致命错误、错误和警告，并显示在信息窗口。

大多数致命错误是指一个内部的编译程序错误。发生此类错误时，编译立即停止，应重新启动编译程序。

错误一般是在编程时，因语法应用不当引起的，根据报错指示，修正一下源程序，重新编译即可。

警告，是指被怀疑的情况，可暂时不去管它。

本附录只给出经常出现的错误信息，供参考。

1. Array bounds missing] in function 函数名

数组缺少右方括号。

2. Array size too large in function 函数名

数组元素定义得太大。

3. Bad file name format in include directive

在宏控制行中，无文件名或缺左引号(或尖括号)。

4. Bad ifdef(或 ifndef，或 undef)directive syntax

条件编译控制行中，无标识符。

5. Both return and return of a value used

有与 return 不一致的 return 语句。

6. Case outside of switch

无 switch 关键字或 case 出现在 switch 语句的外边。

7. Case statement missing :

case 的常量表达式后者无冒号。

8. Character constant too long

字符常量太长。

9. Code has no effect in function

多方面原因，如：变量无类型/存储类别说明；指针相加造成；数组或结构未给尺寸；使用了非 c 语句。

10. Compound statement missing }

复合语句少花括号}。

11. Declaration missing;

语句少分号。

12. Declaration need type or storage class

变量无类型/存储类别说明，以及 struct 的类型名后少{等。

13. Declaration syntax error
说明中有语法错误。
14. Default outside of switch
Default 定义在 switch 之外。
15. Define directive needs identifier
#define 控制行少标识符。
16. Do statement must have while
do 语句后面无 while 对应。
17. do-while statement missing(/)/;
do-while 语句少左圆括号或右圆括号或分号。
18. Duplicate case
case 的常量表达式定义重复。
19. Enum syntax error
enum 语法错误。
20. Expression syntax
表达式语法错误或->右边无结构成员。
21. For statement missing(/)/;
for 后面无左圆括号或右圆括号或分号。
22. Function call missing)
函数调用时，缺右圆括号或参数之间不是逗号。
23. Goto statement missing label
goto 语句少标号。
24. If statement missing(/)
if 语句少左或右圆括号。
25. Illegal character'#' (0xxx)
#后不是宏指令名。
26. Illegal octal digit in function
出现非法 8 进制数。
27. Illegal use of floating point
符点数使用场合搞错。
28. Illegal use of point
非法的使用指针。
29. Incorrect number format
数据格式不正确。
30. Incorrect use of default
default 后无冒号。
31. Invalid macro argument separator
无效的宏参数分隔符。
32. Invalid use of arrow
->右边无结构成员。
33. Invalid use of dot

之后无结构成员。

34. Lvalue required

赋值号左边不是变量。

35. Mismatch number of parameters in definition

定义中的参数个数不匹配。

36. Misplaced break

在 switch 外使用了 break

37. Misplaced decimal point

数据的小数点位置不正确。

38. Misplaced else directive

else 伪命令的位置不正确。

39. No declaration for function 'xxxxx'

给出的函数没有说明。

40. No file name ending

没有给出要求的文件名结束符。

41. Non-portable pointer assignment

试图对非移植指针赋值。

42. Not an allowed type

不允许的类型。

43. Out of memory

内存不足。

44. Parameter 'xxxxx' is never used

给出的参数没有使用。

45. Pointer required on left side of operator

操作符左边不是指针。

46. Possible incorrect assignment

可能是不正确的赋值。

47. Possible use of 'xxxx' before definition

可能在定义之前使用了给出的变量。

48. Redefinition of 'xxxx' is not identical

重定义给出的标识符。

49. Size of structure or array not known

结构或数组的长度未定义。

50. Statement missing

语句遗漏了分号。

51. Switch statement missing(/)

switch 语句少左或右圆括号。

52. Too many default cases

switch 语句中 default 多于一个。

53. Too few parameters in call

函数调用中的参数个数不够。

54. Too many types in declaration
说明中数据类型关键字太多。
55. Two consecotive dots
使用了两个句号。
56. Type missmatch in parameter 'xxxxx'
给出的参数类型不匹配。
57. Unable to open include/input file 'xxxxx'
打不开给出的有关文件。
58. Undefined label/structure/symbol 'xxxxx'
指定的标号或结构或符号未定义。
59. Unexpected end of file in comment started on line n
n 行的注释少*/。
60. Unknown preprocessor directive 'xxx'
给出的预处理宏指令不能识别。
61. Unreachable code
出现不能检索的源码。
62. Unterminated string or character constant
字符或字符串常量少右引号。
63. Void function may not return a value
void 函数不能返回值。
64. While statement missing(/)
while 语句少左或右圆括号。
65. Wrong number of arguments in of 'xxxxx'
调用宏时，参数个数不对。
66. 'xxxx' declared but never used
没有使用定义了的此变量。
67. 'xxxx' is assigned a value which is never used
此变量的赋值一直未使用。
68. 'xxxx' not an argument
给出生的项不是函数参数。
69. 'xxxx' not part of structure
给出的项不是结构的一部分。
70. Zero length structure
结构的长度为零。